AMSTRAD

ROUTINES EN ASSEMBLEUR

JEAN-CLAUDE DESPOINE

SYBEX

AMSTRAD
ROUTINES EN ASSEMBLEUR

DANS LA MÊME COLLECTION

Amstrad jeux d'action, P. Monsaut
Amstrad premiers programmes, R. Zaks
Amstrad 56 programmes, S.R. Trost
Amstrad exploré, J. Braga
Amstrad programmation en assembleur, G. Fagot-Barraly
Amstrad guide du graphisme, J. Wynford
Amstrad CP/M 2.2, A. d'Hardancourt
Amstrad astrologie, numérologie, biorythmes, P. Bourgault
Amstrad graphisme en trois dimensions, T. Lachant-Robert
Amstrad Multiplan, Amstrad
Amstrad CP/M plus, A. d'Hardancourt
Amstrad Astrocalc, G. Blanc/P. Destrebecq
Amstrad gagnez aux courses, J.-C. Despoine
Amstrad créer de nouvelles instructions, J.-C. Despoine
Amstrad Locoscript, B. Le Dû
Amstrad mise au point des programmes BASIC, C. Vivier/Y. Jacob
Amstrad mieux programmer en assembleur, T. Lachant-Robert
Amstrad jeux de réflexion, G. Fagot-Barraly
Amstrad jeux en assembleur, E. Ravis
Amstrad techniques de programmation des jeux, G. Fagot-Barraly
Amstrad Logo, A. d'Hardancourt *(à paraître)*
Amstrad programmes en langage machine, S. Webb *(à paraître)*
Amstrad guide du DOS, Amstrad *(à paraître)*
Amstrad introduction à la programmation en assembleur du Z80, A. d'Hardancourt *(à paraître)*
Amstrad systèmes d'exploitation, Amstrad *(à paraître)*

Jean-Claude DESPOINE

AMSTRAD
ROUTINES EN ASSEMBLEUR

Paris • Berkeley • Düsseldorf

ISBN : 2-7361-0203-7

■ SOMMAIRE

■ AVANT-PROPOS

Destiné aux possesseurs de 464, 664 et 6128, cet ouvrage propose une vingtaine de routines originales en langage machine, susceptibles d'en améliorer considérablement les performances.

Les différents programmes présentés sont d'autre part prétexte à explorer l'organisation interne de ces machines et à en étudier le fonctionnement. Entre autres sujets traités, citons par exemple et en vrac :

- Les techniques de détournement des vecteurs d'appel de routines internes ou des vecteurs d'indirection.
- La programmation des interruptions.
- La programmation des sons en langage machine.
- L'écriture en mémoire écran et les animations.
- La lecture en mémoire ROM et l'utilisation des instructions de RESTART.
- La table du générateur de caractères.
- Le rangement en RAM d'un programme BASIC.
- L'organisation des tableaux.
- L'intégration de nouvelles instructions par RSX.
- Les variables système, etc.

Un explicatif accompagne chacune des routines, ainsi qu'un programme de chargement BASIC et de démonstration ; il convient toutefois de préciser que le lecteur est supposé connaître le minimum de base concernant les instructions du Z80.

Nous avons néanmoins pensé aux débutants, et les listings assembleur sont présentés d'une manière qui, si elle n'est pas très orthodoxe, a le mérite d'être plus aisément compréhensible (les numéros de ligne, en particulier, ne sont là que pour faciliter les explications). Les vieux routiers de l'assembleur voudront bien nous le pardonner et ne devraient pas, en tout état de cause, en être gênés.

Les routines sont systématiquement implantées à partir de l'adresse 40000 ; pour en faciliter le déplacement éventuel, toutes les adresses absolues internes sont signalées par une étoile (*).

Les programmes fonctionnent tels quels sur le 464, et les lignes à modifier sur le 664 et le 6128 sont signalées par un dièse (#).

Généralement, seuls quelques octets diffèrent, et les informations spécifiques à ces deux machines sont fournies à chaque fin de chapitre.

*
* *

Nous espérons naturellement que cet ouvrage vous ouvrira toutes grandes les portes de la programmation en langage machine, qui n'est pas aussi complexe que son aspect quelque peu rébarbatif pourrait le laisser penser. Quoi qu'il en soit, le travail que nécessite sa maîtrise est un excellent investissement, eu égard aux extraordinaires possibilités qu'il offre.

Un conseil, enfin : le langage machine est beaucoup moins tolérant que le BASIC en ce qui concerne les erreurs. Un chiffre tapé de travers, un paramètre erroné, et vous risquez de voir se produire sur l'écran des choses parfois ahurissantes, mais rarement agréables en définitive. Dès que des routines machine sont en jeu, n'hésitez donc pas à faire des sauvegardes fréquentes de vos programmes, et systématiquement avant tout essai.

Bon voyage à l'intérieur du CPC !

1 ÉCRITURE VERTICALE

Il est parfois nécessaire, lors de présentations d'écran particulières, d'écrire des chaînes de caractères verticalement (pour annoter des graphiques, par exemple). Cela peut bien sûr se faire en BASIC, sous réserve d'une généreuse utilisation des instructions LOCATE et PRINT, ce qui rend la procédure laborieuse et fastidieuse.

La routine proposée ci-après simplifie considérablement les choses, puisque après son appel tous les caractères sortis par PRINT s'affichent les uns en dessous des autres, sans qu'il soit nécessaire d'intervenir autrement.

Son format d'appel est tout simplement CALL 40000, et un appel à l'adresse 40012 remet l'écriture horizontale.

```
1     40000  215ABB    LD HL,&BB5A
2     40003  36C3      LD (HL),&C3
3     40005  23        INC HL
4     40006  3658      LD (HL),&58    *
5     40008  23        INC HL
6     40009  369C      LD (HL),&9C    *
7     40011  C9        RET

8     40012  215ABB    LD HL,&BB5A
9     40015  36CF      LD (HL),&CF
10    40017  23        INC HL
11    40018  3600      LD (HL),&00    #
12    40020  23        INC HL
13    40021  36        LD (HL),&94    #
14    40023  C9        RET

15    40024  CD669C    CALL 40038     *
16    40027  3E08      LD A,8
17    40029  CD669C    CALL 40038     *
18    40032  3E0A      LD A,10
19    40034  CD669C    CALL 40038     *
20    40037  C9        RET

21    40038  CF0094    RST 8          #
22    40041  C9        RET
```

Tout le programme repose en fait sur le détournement d'un vecteur d'appel du système d'exploitation, en l'occurrence &BB5A. Ce procédé sera utilisé dans d'autres programmes, et il est peut-être bon d'en rappeler brièvement le principe :

Vous savez certainement déjà qu'il est possible d'accéder, grâce à ces vecteurs d'appel, aux routines de la ROM d'exploitation du CPC (chacun connaît par exemple le CALL &BB06 qui provoque l'attente d'une frappe de touche).

Dans la plupart des cas, cependant, cette possibilité n'est exploitable qu'à partir d'un programme en langage machine, puisque les registres doivent au préalable être chargés correctement. C'est en particulier le cas pour le vecteur &BB5A, qui permet la sortie sur écran d'un caractère : avant son appel, l'accumulateur doit contenir le code ASCII du caractère souhaité (signalons que si ce code est compris entre 0 et &1F, donc s'il s'agit d'un code de contrôle, il sera traité comme tel).

Ces vecteurs d'appel sont composés, en mémoire RAM, de trois octets. Voyons le cas du vecteur &BB5A :

```
PEEK(&BB5A) = &CF
PEEK(&BB5B) = &00
PEEK(&BB5C) = &94
```

Attention !

Les valeurs ci-dessus sont celles du 464. Pour le 664 et le 6128, voir en fin de chapitre (de toute façon, le raisonnement reste identique).

Le premier octet consiste donc en un RESTART 8, et les deux suivants fournissent une adresse ROM codée d'une manière un peu particulière. Nous n'entrerons pas ici dans les détails sur l'habile utilisation, par les concepteurs du CPC, des instructions de RESTART. Considérons simplement que ce RESTART 8 n'est finalement qu'un RESTART à l'adresse indiquée par les deux octets suivants. Concernant cette adresse, il faut signaler que les bits 14 et 15 ne doivent pas être pris en compte. Selon qu'ils sont à 1 ou à 0, ces deux bits ne servent en effet qu'à signaler que l'adresse fournie doit être recherchée soit en ROM, soit en RAM. C'est ainsi par exemple que la valeur &9400 indique en fait l'adresse &1400 de la ROM d'exploitation.)

Quoi qu'il en soit, une connaissance approfondie du mécanisme des RESTART n'est pas indispensable pour effectuer des détournements judicieux des vecteurs d'appel. Dans notre programme, ce détournement est réalisé par les lignes 1 à 7. Dorénavant, lorsque le système interne va vouloir utiliser le vecteur &BB5A pour afficher un caractère, il rencontrera, au lieu du RST normal, un JUMP à l'adresse &9C58 (40024). C'est bien sûr à cette adresse qu'est installée notre routine de remplacement.

Cette nouvelle routine commence par appeler le sous-programme 40038, qui est l'exacte réplique du RST original. Le caractère contenu dans A est donc affiché normalement. Le programme se poursuit ensuite en ligne 16, où l'accumulateur est chargé avec la valeur 8, qui est un code de contrôle ayant pour effet de déplacer le curseur vers la gauche. Ce code est exécuté, puis A est chargé avec la valeur 10, code de contrôle qui fait descendre le curseur d'une ligne. Le curseur est donc maintenant positionné sous le caractère précédent, et c'est à cet endroit que sera affiché le caractère suivant.

La remise en état du vecteur &BB5A (lignes 8 à 14) s'effectue tout naturellement en remettant les valeurs initiales dans (&BB5A), (&BB5B) et (&BB5C).

Voici maintenant le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 3
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 215ABB36C3233658233369CC921,4BF
170 DATA 5ABB36CF233600233694C9CD66,55C
180 DATA 9C3E08CD669C3E0ACD669CC9CF,660
```

```
190 DATA 0094C9,15D
200 '
210 '
220 '           DEMONSTRATION
230 '
240 CALL 40000
250 CLS:LOCATE 20,2
260 PRINT"ECRITURE VERTICALE"
270 CALL 40012
```

■ ADAPTATION AU 664

Le contenu des emplacements mémoire (&BB5A + 1) et (&BB5A + 2) n'est pas le même :

PEEK(&BB5B) = &FA
PEEK(&BB5C) = &93

Il suffit donc tout simplement de modifier ces valeurs. Dans le programme de chargement BASIC, deux lignes doivent être modifiées :

```
170 DATA 5ABB36CF2336FA233693C9CD66,655
190 DATA FA93C9,256
```

■ ADAPTATION AU 6128

Là encore, l'aspect du vecteur est différent :

PEEK(&BB5B) = &FE
PEEK(&BB5C) = &93

Les lignes suivantes du programme de chargement BASIC doivent être modifiées :

```
170 DATA 5ABB36CF2336FE233693C9CD66,659
190 DATA FE93C9,25A
```

2 ÉCRITURE ALTERNÉE

Cette routine, qui comme la précédente est basée sur le détournement du vecteur d'écriture &BB5A, permet de sortir des chaînes dont les caractères sont alternativement de la couleur 1, 2 et 3. Cela permet des présentations originales et des effets spéciaux intéressants, tel que celui qui est proposé en démonstration.

L'installation de l'écriture alternative se fait par CALL 40000, et le retour à l'écriture normale par CALL 40012.

Voici le listing assembleur :

```
1     40000   215ABB     LD HL,&BB5A
2     40003   36C3       LD (HL),&C3
3     40005   23         INC HL
4     40006   365D       LD (HL),&5D    *
5     40008   23         INC HL
6     40009   369C       LD (HL),&9C    *
7     40011   C9         RET

8     40012   3E01       LD A,1
9     40014   CD90BB     CALL &BB90
10    40017   215ABB     LD HL,&BB5A
11    40020   36CF       LD (HL),&CF
12    40022   23         INC HL
13    40023   3600       LD (HL),0      #
14    40025   23         INC HL
15    40026   3694       LD (HL),&94    #
16    40028   C9         RET

17    40029   F5         PUSH AF
18    40030   E5         PUSH HL
19    40031   CD93BB     CALL &BB93
20    40034   FE01       CP 1
21    40036   2004       JRNZ 40042
22    40038   3E02       LD A,2
23    40040   180A       JR 40052
24    40042   FE02       CP 2
25    40044   2004       JRNZ 40048
```

```
26    40046   3E03        LD A,3
27    40048   1802        JR 40052
28    40050   3E01        LD A,1

29    40052   CD90BB      CALL &BB90
30    40055   E1          POP HL
31    40056   F1          POP AF
32    40057   CF0094      RST 8           #
```

Nous ne reviendrons pas sur le détournement effectué par les lignes 1 à 7, et dont le processus vous est maintenant familier. Il renvoie à l'adresse 40029 où est installée la routine de remplacement. Pour obtenir l'alternance entre les couleurs de code 1, 2 et 3, il suffit d'établir le fonctionnement suivant : avant chaque sortie d'un caractère la couleur de stylo est testée. Si elle vaut 1, il faudra installer la couleur 2. Si elle vaut 2, il faudra installer la couleur 3. Si enfin elle vaut 3, ce sera la couleur 1. Et ainsi de suite...

Les registres AF et HL sont tout d'abord préservés. Cela est absolument indispensable puisque, lorsque le programme interne utilise le vecteur &BB5A, les données dont il a besoin pour afficher le caractère y sont rangées (code du caractère et situation du curseur) et qu'il faudra en fin de compte les restituer.

La couleur actuelle du stylo est ensuite testée grâce à l'appel de &BB93. Au retour, le code de cette couleur est dans l'accumulateur. En fonction de ce code, la valeur 1, 2 ou 3 est chargée dans A, après quoi le programme se poursuit en ligne 29. Là, l'appel de &BB90 fixe la couleur choisie, les contenus initiaux des registres HL et A sont récupérés, puis le RESTART 8 est exécuté.

En partant de ce modèle, il est très facile de réaliser une écriture alternée avec un nombre de couleurs plus ou moins important.

Voici le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 4
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
```

```
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 215ABB36C323365D23369CC93E,4E1
170 DATA 01CD90BB215ABB36CF23360023,4D0
180 DATA 3694C9F5E5CD93BBFE0120043E,6E9
190 DATA 02180AFE0220043E0318023E01,1E2
200 DATA CD90BBE1F1CF0094,54D
210 '
220 '
230 '          DEMONSTRATION
240 '
250 INK 0,13:INK 1,2:INK 2,26:INK 3,6
260 CLS:CALL 40000
270 LOCATE 2,12
280 PRINT"VOICI UNE ROUTINE TRES PATRIOT
IQUE..."
290 FOR I=1 TO 10
300 INK 1,26:INK 2,6:INK 3,2
310 FOR T=1 TO 300:NEXT T
320 INK 1,6:INK 2,2:INK 3,26
330 FOR T=1 TO 300:NEXT T
340 INK 1,2:INK 2,26:INK 3,6
350 FOR T=1 TO 300:NEXT T
360 NEXT I
370 REP$=INKEY$:IF REP$="" THEN 370
380 INK 0,1:INK 1,24:INK 2,20:INK 3,6
390 CALL 40012
```

Comme vous pourrez le voir après lancement, l'association de cette routine et de l'instruction BASIC INK donne des résultats assez plaisants...

■ ADAPTATION AU 664 ET AU 6128

Là encore, la seule modification se rapporte à l'adresse qui suit le RESTART, et trois lignes du programme BASIC doivent être modifiées :

Pour le 664

```
170 DATA 01CD90BB215ABB36CF2336FA23,5CA
180 DATA 3693C9F5E5CD93BBFE0120043E,6E8
200 DATA CD90BBE1F1CFFA93,646
```

Pour le 6128

```
170 DATA 01CD90BB215ABB36CF2336FE23,5CE
180 DATA 3693C9F5E5CD93BBFE0120043E,6E8
200 DATA CD90BBE1F1CFFE93,64A
```

3 ÉCRITURE FAÇON TÉLÉSCRIPTEUR

Cette routine provoque l'émission d'un son entre l'affichage de chacun des caractères d'une chaîne. Cela donne effectivement une impression de téléscripteur et convient fort bien, dans le déroulement d'un programme, à l'affichage de titres ou de messages importants.

D'ores et déjà, vous devez vous douter que c'est une fois de plus ce malheureux vecteur &BB5A qui va en faire les frais et qui va être détourné.

Cette écriture particulière est installée par un CALL 40000, et le retour à la normale se fait par CALL 40012.

Plusieurs méthodes étaient possibles pour fabriquer ce programme, dont la plus simple était bien sûr d'utiliser le code de contrôle 7 pour sortir un coup de cloche entre chaque affichage de caractère. Nous avons néanmoins préféré un système plus complexe, mais permettant d'utiliser les nombreuses possibilités sonores du CPC. D'autre part, cela nous permet d'aborder les techniques de programmation des sons à partir du langage machine.

Voici le listing assembleur :

```
1     40000   215ABB     LD HL,&BB5A
2     40003   36C3       LD (HL),&C3
3     40005   23         INC HL
4     40006   3658       LD (HL),&58    *
5     40008   23         INC HL
6     40009   369C       LD (HL),&9C    *
7     40011   C9         RET

8     40012   215ABB     LH HL,&BB5A
9     40015   36CF       LD (HL),&CF
10    40017   23         INC HL
11    40018   3600       LD (HL),0      #
12    40020   23         INC HL
13    40021   3694       LD (HL),&94    #
14    40023   C9         RET

15    40024   F5         PUSH AF
```

```
16    40025   E5          PUSH HL
17    40026   D5          PUSH DE
18    40027   218C3C      LD HL,15500
19    40030   2B          DEC HL
20    40031   7C          LD A,H
21    40032   B5          OR L
22    40033   20FB        JRNZ 40030
23    40035   21709C      LD HL,40048     *
24    40038   CDAABC      CALL &BCAA
25    40041   D1          POP DE
26    40042   E1          POP HL
27    40043   F1          POP AF

28    40044   CF0094      RST 8           #
29    40047   C9          RET

30    40048   01          Queue sonore
31    40049   00
32    40050   00
33    40051   82
34    40052   00
35    40053   00
36    40054   0C
37    40055   05
38    40056   00
```

La routine de remplacement est installée à partir de l'adresse 40024 et commence par préserver les registres AF, DE et HL. Ce dernier est ensuite chargé avec une valeur arbitraire, en prévision des lignes 19 à 22, qui représentent une simple boucle de temporisation (HL est décrémenté à chaque passe et le OR L de la ligne 21 n'a pour résultat 0 que si H *et* L valent tous deux 0, donc si HL = 0. Ces x décrémentations représentent donc un délai d'exécution qui dépend naturellement du chargement initial de HL. Cela est tout à fait comparable au procédé BASIC de temporisation avec FOR et NEXT).

Cette temporisation est indispensable pour la raison suivante : la programmation d'un son en langage machine est, pour ainsi dire, instantanée ; le programme range les différents éléments qui constituent ce son dans une file d'attente prévue à cet effet (sous réserve qu'il y ait encore de la place disponible ; nous reviendrons sur ce problème dans un autre chapitre), après quoi il ne s'en préoccupe plus et poursuit en séquence.

Imaginons maintenant que cette temporisation n'existe pas dans notre programme et que vous vouliez sortir — façon téléscripteur — une chaîne de cinq caractères. En une fraction de seconde, la chaîne serait affichée, alors même que le premier des cinq sons serait à peine en cours de traitement (c'est-à-dire en train de sortir du haut-parleur) et que les quatre autres seraient encore installés dans la file, en train d'attendre patiemment leur tour. Le résultat serait donc l'émission d'un bruit continu (aucune interruption ne serait perceptible entre chaque son) qui sortirait bien longtemps (relativement) après que la chaîne aurait été affichée. L'effet obtenu serait donc plutôt du genre sirène à retardement...

Il est par conséquent nécessaire qu'une temporisation ait lieu d'une manière ou d'une autre (nous verrons plus loin qu'il existe une autre solution), pour laisser à chaque son le temps d'être traité avant de passer au caractère suivant.

Le registre HL est ensuite chargé, en ligne 23, avec l'adresse du premier octet d'une queue sonore qui doit en contenir neuf, dont les significations sont les suivantes :

Octet 1 numéro du canal.
Octet 2 enveloppe de volume (si elle existe).
Octet 3 enveloppe de ton (si elle existe).
Octet 4 période (octet faible).
Octet 5 période (octet fort).
Octet 6 bruitage éventuel.
Octet 7 volume initial.
Octet 8 octet faible de la durée (en centièmes de seconde).
Octet 9 octet fort de la durée.

Pour plus de précisions concernant le rôle de ces différents paramètres, nous vous renvoyons au manuel d'utilisation de votre machine dans lequel tout cela est expliqué en détail. Rien ne valant l'expérience pratique, vous pouvez également essayer de modifier l'une ou l'autre valeur de la queue sonore du programme et constater les résultats.

HL étant correctement chargé, il suffit maintenant d'appeler le vecteur &BCAA, qui va installer notre queue sonore pour traitement. Les trois registres sont ensuite récupérés et le RESTART de sortie de caractère est effectué.

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 4
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 215ABB36C323365823369CC921,4BF
170 DATA 5ABB36CF233600233694C9F5E5,603
180 DATA D5218C3C2B7CB520FB21709CCD,62F
190 DATA AABCD1E1F1CF0094C901000082,6B8
200 DATA 00000C0500,11
210 '
220 '          DEMONSTRATION
230 '
240 CLS:CALL 40000
250 LOCATE 8,10
260 PRINT"VOICI UNE DEMONSTRATION"
270 LOCATE 6,12
280 PRINT"DE CETTE ETRANGE ROUTINE..."
290 CALL 40012
```

■ ADAPTATION AU 664 ET AU 6128

Pour l'un comme pour l'autre, deux lignes du programme de chargement BASIC doivent être modifiées.

Pour le 664

```
170 DATA 5ABB36CF2336FA233693C9F5E5,6FC
190 DATA AABCD1E1F1CFFA93C901000082,7B1
```

Pour le 6128

```
170 DATA 5ABB36CF2336FE233693C9F5E5,700
190 DATA AABCD1E1F1CFFE93C901000082,7B5
```

4 MISE EN ÉVIDENCE DE ZONES DE LISTING

Lorsque l'on travaille sur un programme BASIC de longueur importante, une des choses les plus éprouvantes est la recherche d'une zone précise du programme par listing d'écran. Rien de tel pour fatiguer les yeux et l'esprit que de tenter le repérage d'une ligne particulière dans cette multitude de lettres et de chiffres qui défilent.

La solution peut être l'utilisation massive de lignes de REM, avec des titres mis en évidence par des étoiles, des dièses, et Dieu sait quoi encore. Solution bien archaïque, et c'est pourquoi nous vous proposons maintenant la simulation en langage machine d'une instruction existant sur d'autres ordinateurs et qui permet, lors de listings d'écran, l'affichage en couleur de code 3 ou en vidéo inverse de zones délimitées. Le processus est le suivant :

Le vecteur &BB5A (encore lui...) est détourné, de telle manière que si le caractère "[" est rencontré, la couleur de stylo est mise à 3. Inversement, si le caractère "]" est rencontré, la couleur de stylo est remise à 1. Il est donc très facile de faire ressortir des zones de programme en les bornant par ces deux caractères. L'installation du détournement se fait par CALL 40000 et la remise en état par CALL 40012.

Remarque

Si le programme lui-même, en cours d'exécution, doit sortir un des deux caractères évoqués plus haut, le changement de couleur aura également lieu. En tout état de cause, il est très facile de choisir un autre caractère de délimitation.

Voici le listing assembleur :

```
1     40000   215ABB     LD HL,&BB5A
2     40003   36C3       LD (HL),&C3
3     40005   23         INC HL
4     40006   3658       LD (HL),&58    *
5     40008   23         INC HL
6     40009   369C       LD (HL),&9C    *
7     40011   C9         RET

8     40012   215ABB     LD HL,&BB5A
9     40015   36CF       LD (HL),&CF
```

```
10    40017   23          INC HL
11    40018   3600        LD (HL),&00    #
12    40020   23          INC HL
13    40021   36          LD (HL),&94    #
14    40023   C9          RET

15    40024   F5          PUSH AF
16    40025   E5          PUSH HL
17    40026   FE5B        CP &5B
18    40028   2007        JRNZ 40037
19    40030   3E03        LD A,3
20    40032   CD90BB      CALL &BB90
21    40035   1809        JR 40046
22    40037   FE5D        CP &5D
23    40039   2005        JRNZ 40046
24    40041   3E01        LD A,1
25    40043   CD90BB      CALL &BB90

26    40046   E1          POP HL
27    40047   F1          POP AF
28    40048   CF0094      RST 8          #
```

Rien de bien particulier à signaler : la routine de détournement est installée en 40024 ; le caractère qui doit être affiché (et qui est contenu dans A) est comparé (ligne 17) au code ASCII de "[" (&5B) ou de "]" (&5D). S'il ne s'agit ni de l'un ni de l'autre, le programme saute en ligne 26 où le caractère est sorti normalement. S'il s'agit de "[", la couleur d'écriture est mise à 3, avant la sortie du caractère, par l'appel de &BB90, et enfin s'il s'agit de "]" la couleur est remise à 1.

Pour obtenir une vidéo inverse au lieu d'un changement de couleur, il suffit de remplacer le vecteur &BB90 par le vecteur &BB9C, qui inverse les couleurs de stylo et de papier. Aucun registre ne doit être chargé d'une manière particulière avant son appel, et les octets 40030, 40031, 40041 et 40042 peuvent donc être mis à 0 (cela n'est pas obligatoire).

Voici le listing de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
```

```
50 AD=40000
60 FOR I=0 TO 3
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 215ABB36C323365823369CC921,4BF
170 DATA 5ABB36CF233600233694C9F5E5,603
180 DATA FE5B20073E03CD90BB1809FE5D,555
190 DATA 20053E01CD90BBE1F1CF0094,5B1
200 '
210 '
220 '[ IL EST POSSIBLE DE FAIRE
230 '  RESSORTIR DES REM. OU DES
240 '  ZONES DE PROGRAMMES : ]
250 '
260 '[:w=0:if w=1 then end:']
270 '
280 CALL 40000:LIST
```

■ ADAPTATION AU 664 ET AU 6128

Comme d'habitude, deux lignes sont à modifier dans le programme de chargement BASIC :

Pour le 664

```
170 DATA 5ABB36CF2336FA233693C9F5E5,6FC
190 DATA 20053E01CD90BBE1F1CFFA93,6AA
```

Pour le 6128

```
170 DATA 5ABB36CF2336FE233693C9F5E5,700
190 DATA 20053E01CD90BBE1F1CFFE93,6AE
```

5 TRON EN FENÊTRE 7

Ce cinquième programme, permettant une parfaite maîtrise de l'instruction TRON, terminera la série de ceux qui utilisent le système de détournement du vecteur &BB5A.

S'il est vrai que l'instruction BASIC TRON est infiniment précieuse pour la traque des *bugs* de programmation, il faut également avouer qu'elle manque un peu de finesse : les numéros de ligne s'affichent n'importe où sur l'écran, au beau milieu des affichages spécifiques du programme, et il est souvent bien difficile de s'y retrouver. Le programme suivant permet la sortie des numéros d'une manière plus "raisonnable", puisqu'ils ne s'afficheront qu'en fenêtre 7 (celle-ci doit bien entendu avoir été définie auparavant). Nous y avons d'autre part intégré une autre fonction utile, puisqu'il suffit d'appuyer sur la barre d'espace pour interrompre provisoirement l'exécution du programme. Dès relâchement de la touche, le programme se poursuit normalement.

L'installation de la routine de détournement se fait par CALL 40000 et la remise en l'état initial par CALL 40012.

Le principe n'est pas sans rappeler celui du programme précédent. Comme vous le savez, en effet, l'affichage des numéros de ligne par TRON est toujours précédé de "[" et suivi de "]". D'autre part, un simple coup d'œil au listing de la zone de ROM BASIC consacrée à l'instruction TRON permet de constater que le vecteur &BB5A y est utilisé.

Il suffira donc, lorsque "[" sera rencontré, d'installer la fenêtre 7 avant d'autoriser la sortie du numéro de ligne. Cette fenêtre 7 a été choisie arbitrairement et il est naturellement possible d'en utiliser une autre. (Attention, le canal 8, celui de l'imprimante, ne peut être sélectionné sans aménagements particuliers.)

Voici le listing assembleur :

```
1     40000  215ABB     LD HL,&BB5A
2     40003  36C3       LD (HL),&C3
3     40005  23         INC HL
4     40006  3658       LD (HL),&58    *
5     40008  23         INC HL
```

```
6     40009  369C      LD (HL),&9C    *
7     40011  C9        RET

8     40012  215ABB    LD HL,&BB5A
9     40015  36CF      LD (HL),&CF
10    40017  23        INC HL
11    40018  3600      LD (HL),&00    #
12    40020  23        INC HL
13    40021  36        LD (HL),&94    #
14    40023  C9        RET

15    40024  E5        PUSH HL
16    40025  F5        PUSH AF
17    40026  FE5B      CP &5B
18    40028  200C      JRNZ 40042
19    40030  3E2F      LD A,&2F
20    40032  CD1EBB    CALL &BB1E
21    40035  20F9      JRNZ 40030
22    40037  3E07      LD A,7
23    40039  CDB4BB    CALL &BBB4

24    40042  F1        POP AF
25    40043  E1        POP HL
26    40044  CF0094    RST 8          #
```

La routine de remplacement, installée en 40024, vérifie si le caractère qui doit être sorti est "[". Si tel n'est pas le cas, un saut est effectué en ligne 24 où le caractère est sorti normalement. Si par contre le caractère correspond, le programme exécute les lignes 19 à 23 :

L'accumulateur est tout d'abord chargé avec le code de la barre d'espace (&2F), puis le vecteur &BB1E est sollicité. Au retour, l'indicateur de zéro (Z) est vrai *si et seulement si* la touche testée *n'est pas* pressée.

Si la barre d'espace est pressée, le programme boucle des lignes 21 à 29. Si elle n'est pas pressée, le numéro de fenêtre est chargé dans A, avant l'appel de &BBB4 qui installe cette fenêtre. Le caractère est ensuite sorti normalement.

Une chose peut paraître étrange : à aucun moment, en effet, la fenêtre courante (c'est-à-dire celle utilisée par le programme) n'est réinstallée. On pourrait donc s'imaginer qu'à partir de la première ren-

contre de "[" tout sera écrit en fenêtre 7, y compris ce qui sera envoyé par le programme.

Il n 'y a en réalité aucun souci à se faire. A chaque rencontre de l'instruction PRINT, le logiciel interne teste dans quelle fenêtre doit être écrit *l'ensemble de la chaîne de caractères qui suit*, la fenêtre 0 étant prise par défaut (si aucun canal n'est précisé).

Cela se passe de la même manière lorsque nous installons la fenêtre 7 avant la sortie d'un numéro de ligne : à ce moment-là, le système "sait" déjà que l'ensemble "xxxxx" forme une seule chaîne qui doit être affichée intégralement dans la même fenêtre.

Voici le listing de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 3
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 215ABB36C3233658233369CC921,4BF
170 DATA 5ABB36CF233600233694C9E5F5,603
180 DATA FE5B200C3E2FCD1EBB20F93E07,4F6
190 DATA CDB4BBF1E1CF0094,571
200 '
210 '
220 '            DEMONSTRATION
230 '
240 WINDOW#7,1,40,24,25
250 PAPER#7,3:CLS#7:CALL 40000
260 '
270 TRON
280 FOR I=1 TO 20
290 REM
```

```
300 PRINT"ESSAI ";
310 REM
320 NEXT I
330 '
340 CALL 40012:TROFF
```

■ ADAPTATION AU 664 ET AU 6128

Deux lignes sont à modifier :

Pour le 664

```
170 DATA 5ABB36CF2336FA233693C9E5F5,6FC
190 DATA CDB4BBF1E1CFFA93,66A
```

Pour le 6128

```
170 DATA 5ABB36CF2336FE233693C9E5F5,700
190 DATA CDB4BBF1E1CFFE93,66E
```

6 CENTRAGE DE CHAÎNE

Nous allons maintenant nous attaquer à un programme un peu plus important qui effectue le centrage "automatique" de chaînes sur l'écran. Cela vous permettra donc d'en finir une fois pour toutes avec les pénibles opérations sur feuilles quadrillées que, comme tout le monde, vous êtes certainement obligés d'utiliser pour préparer vos affichages.

Le format d'appel de cette routine est le suivant :

CALL 40000,NL,@X$,NF

- Le paramètre **NF**, indiquant sur quel canal (fenêtre) doit être sortie la chaîne, est optionnel. S'il est omis, la fenêtre 0 est prise par défaut.
- Le paramètre **NL**, obligatoire, représente le numéro de ligne.
- Le paramètre **@X$**, obligatoire lui aussi, représente l'adresse du descripteur de la chaîne X$, qui doit bien entendu être définie avant l'appel.

Rappelons brièvement que la fonction du pointeur de variable (@) est de renvoyer l'adresse RAM où est rangée une variable, qu'il s'agisse d'une variable numérique ou alphanumérique. Dans le cas d'une variable numérique, cependant, l'adresse fournie n'est pas directement celle de la variable mais celle de son descripteur de chaîne, composé de trois octets :

Octet 1 longueur de la chaîne (entre 1 et 255).
Octet 2 adresse de la chaîne (octet faible).
Octet 3 adresse de la chaîne (octet fort).

Pour ceux qui l'ignoreraient, il convient peut-être également de signaler que lorsque l'appel d'un programme machine est accompagné de paramètres :

1. Les paramètres (dont le nombre ne doit pas excéder 32) sont rangés en mémoire les uns à la suite des autres, en commençant par

le dernier, et le registre IX pointe sur la première adresse de ce bloc (donc sur l'octet faible du dernier paramètre).

2. Le registre DE contient le dernier paramètre.
3. L'accumulateur contient le nombre de paramètres transmis.

Dans ces conditions, il est évident qu'un paramètre optionnel pose un problème particulier... résolu dans notre cas par les lignes 1 à 9 du listing assembleur suivant :

```
1     40000  FE03      CP 3
2     40002  200A      JRNZ 40014
3     40004  7B        LD A,E
4     40005  CDB4BB    CALL &BBB4
5     40008  DD23      INC IX
6     40010  DD23      INC IX
7     40012  1803      JR 40017
8     40014  FE02      CP 2
9     40016  C0        RET NZ

10    40017  3A89B2    LD A,(&B289)   #
11    40020  47        LD B,A
12    40021  3A8BB2    LD A,(&B28B)   #
13    40024  90        SUB A,B
14    40025  3C        INC A
15    40026  DD6601    LD H,(IX+1)
16    40029  DD6E00    LD L,(IX+0)
17    40032  96        SUB A,(HL)
18    40033  3821      JRC 40068
19    40035  E5        PUSH HL
20    40036  2600      LD H,0
21    40038  6F        LD L,A
22    40039  2B        DEC HL
23    40040  E5        PUSH HL
24    40041  29        ADD HL,HL
25    40042  BD        CP L
26    40043  E1        POP HL
27    40044  38F9      JRC 40039
28    40046  23        INC HL
29    40047  65        LD H,L
30    40048  DD6E02    LD L,(IX+2)
31    40051  CD75BB    CALL &BB75
```

```
32    40054   E1          POP HL
33    40055   46          LD B,(HL)
34    40056   23          INC HL
35    40057   5E          LD E,(HL)
36    40058   23          INC HL
37    40059   56          LD D,(HL)
38    40060   EB          EX HL,DE
39    40061   7E          LD A,(HL)
40    40062   CD5ABB      CALL &BB5A
41    40065   23          INC HL
42    40066   10F9        DJNZ 40061

43    40068   AF          XOR A
44    40069   CDB4BB      CALL &BBB4
45    40072   C9          RET
```

Si la comparaison de la ligne 1 est différente de 0, c'est donc en principe que deux paramètres seulement ont été transmis et qu'il n'y a pas à se préoccuper de la fenêtre. Le saut est alors effectué en ligne 8. Là, le nombre de paramètres transmis est comparé à 2. S'il n'est pas non plus égal à ce nombre, c'est donc qu'une erreur a été faite dans le format d'appel de la routine et la main est immédiatement rendue au BASIC. Sinon, le programme, *qui a été conçu comme s'il ne devait jamais y avoir que deux paramètres*, se poursuit normalement.

Voyons maintenant ce qui se passe après la ligne 1 si trois paramètres ont été transmis.

Le numéro de fenêtre étant chargé dans DE (et plus exactement dans E, puisqu'il s'agit forcément d'un nombre inférieur à 255) est chargé dans l'accumulateur et la fenêtre est installée grâce à &BBB4 (vu précédemment). Rien de bien nouveau dans tout cela. Ce qui est plus important, c'est que le registre IX est décrémenté deux fois et se retrouve donc maintenant pointé de la même manière que si deux paramètres seulement avaient été transmis. Il est donc possible maintenant de passer à la partie commune du programme, en ligne 10.

Deux variables système sont utilisés dans ce programme :

- La première, rangée à l'adresse &B289, contient le numéro de la première colonne de la fenêtre courante.
- La seconde, rangée à l'adresse &B28B, contient le numéro de la dernière colonne de la fenêtre courante.

Ces deux adresses sont différentes sur le 664 et le 6128 (voir en fin de chapitre).

Pour effectuer un centrage de chaîne sur écran, il faut :

1. Connaître la largeur de la fenêtre courante (en nombre de colonnes).
2. Connaître la longueur de la chaîne que l'on souhaite afficher.
3. Soustraire cette longueur de la largeur de la fenêtre (si cela est possible).
4. Diviser le résultat par deux.
5. Positionner le curseur en fonction du résultat.
6. Afficher la chaîne.

La première de ces opérations est réalisée par les lignes 10 à 14, qui ne sont pas bien difficiles à comprendre et au terme desquelles A contient la largeur de la fenêtre.

Le registre HL est alors pointé sur l'adresse contenant la longueur de la chaîne (@aX$) qui est soustraite, en ligne 17, de la largeur de la fenêtre. En cas de report, donc si la chaîne est trop longue par rapport à la fenêtre, le programme saute en ligne 43. Là, la fenêtre 0 est installée pour le cas où il y aurait eu un changement de fenêtre au début du programme (le XOR A n'a pas d'autre fonction que de charger A avec 0).

Si la chaîne n'est pas trop longue, le registre HL (qui pointe toujours sur le descripteur de chaîne) est préservé, la valeur xx contenue dans A (= largeur de la fenêtre – longueur de la chaîne) est chargée dans L et H est mis à 0.

La boucle suivante (lignes 22 à 27) a pour objet de charger L avec xx/2 :

Le registre HL est au départ chargé avec xx (en réalité, seul L nous intéresse, puisque xx est inférieur à 255 et tient donc dans le registre de poids faible). Il est décrémenté à chaque passe et l'on vérifie immédiatement si HL*2 est plus grand que xx (grâce au CP L de la ligne 25). Si tel n'est pas le cas, alors HL vaut (xx/2) – 1 et l'opération est terminée après que l'on a corrigé le – 1 en incrémentant HL (ligne 28).

Cette valeur passe ensuite de L dans H, puis L est chargé avec le premier paramètre (numéro de ligne).

En résumé, H contient maintenant le numéro de la colonne où devra s'afficher la chaîne, et L contient le numéro de ligne. Les conditions sont donc remplies pour l'appel du vecteur &BB75 qui fixe le curseur à la position indiquée par H et L.

La chaîne peut à présent être sortie. L'adresse de son descripteur est récupérée sur la pile, et les lignes 33 à 38 ont pour effet de :

1. charger la longueur de chaîne dans B ;
2. charger l'adresse du début de la chaîne d'abord dans DE, ensuite dans HL (ligne 38).

La sortie proprement dite s'effectue simplement grâce à la boucle suivante (lignes 39 à 42) : le caractère sur lequel pointe HL est chargé dans A, le vecteur de sortie de caractère &BB5A est appelé, HL est incrémenté de manière à pointer sur le caractère suivant, et le processus reprend tant que le compteur B n'atteint pas 0.

Pour finir, la fenêtre 0 est réinstallée.

Voici le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 5
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE03200A7BCDB4BBDD23DD2318,5FA
170 DATA 03FE02C03A89B2473A8BB2903C,5C2
180 DATA DD6601DD6E00963821E526006F,4F8
190 DATA 2BE529BDE138F92365DD6E02CD,6AA
200 DATA 75BBE146235E2356EB7ECD5ABB,69C
210 DATA 2310F9AFCDB4BBC9,4E0
220 '
230 '
```

```
240 '           DEMONSTRATION
250 '
260 BORDER 3
270 MODE 1:FOR I=5 TO 21 STEP 2
280 READ A$:CALL 40000,I, @ A$
290 NEXT
300 DATA Voici donc une demonstration de
 cette
310 DATA super routine qui centre
320 DATA automatiquement
330 DATA les chaines
340 DATA de
350 DATA caracteres
360 DATA y compris dans une
370 DATA fenetre precise et fonctionnant
380 DATA bien naturellement dans les 3 m
odes
390 GOTO 390
```

■ ADAPTATION AU 664 ET AU 6128

Les adresses des deux variables système évoquées au cours du chapitre sont identiques sur le 664 et le 6128, mais différentes de celles du 464.

- L'adresse &B289 (colonne de début de la fenêtre courante) devient &B72A.
- L'adresse &B28B (colonne de fin de la fenêtre courante) devient &B72C.

La ligne suivante doit être modifiée dans le listing de chargement BASIC :

```
170 DATA 03FE02C03A2AB7473A2CB7903C,50E
```

7 FIXATION D'UNE FENÊTRE

Un des aspects les plus intéressants du BASIC CPC est probablement la possibilité de définition de fenêtres. On imagine d'ailleurs difficilement, lorsque l'on y a pris goût, d'avoir à utiliser un BASIC qui ne proposerait pas cette option.

Il reste cependant ce que des esprits chagrins, dont nous faisons partie, peuvent considérer comme étant un défaut, ou plutôt comme une lourdeur d'utilisation : l'emploi obligatoire des dièses d'indication de canal. Outre que cela peut parfois allonger sensiblement et rendre confus un programme, on a aussi vite fait de les oublier dans la fièvre de l'écriture.

Vous pouvez désormais considérer ce défaut comme faisant partie du passé, puisque le programme suivant a pour fonction de fixer une fenêtre une fois pour toutes : après son exécution, les instructions LOCATE, PRINT et CLS n'auront plus besoin d'être suivies du dièse de canal.

Quelques restrictions, cependant :

- Cette fixation ne concerne aucune autre instruction (PEN et PAPER devront comme d'habitude être suivis du numéro de canal).
- Sur les 664 et 6128, et tel que le programme est proposé, l'instruction CLS n'est pas fixée non plus. Vous trouverez néanmoins en fin de chapitre les indications nécessaires pour combler cette lacune.

Vous serez peut-être amenés à utiliser de nombreuses fois cette routine dans vos programmes, et c'est pouquoi nous l'avons intégrée en R.S.X. *(Resident System Extension)*. Les formats d'appel seront donc les suivants (rappelons que la barre verticale s'obtient par SHIFT + @) :

|FIXON, NF	Fixe la fenêtre de numéro NF.
|FIXOFF	Supprime la fixation.

Voici le listing assembleur :

```
1     40000   01499C      LD BC,40009    *
2     40003   21549C      LD HL,40020    *
3     40006   C3D1BC      JP &BCD1
4     40009   4E9C        40020          *
5     40011   C3719C      JP 40049       *
6     40014   46          'F'
7     40015   49          'I'
8     40016   58          'X'
9     40017   4F          'O'
10    40018   CE          'N'+&80
11    40019   00          Fin du mot

12    40020   00          4 octets
13    40021   00          Reserves
14    40022   00          KERNEL
15    40023   00

16    40024   01619C      LD BC,40033    *
17    40027   216D9C      LD HL,40045    *
18    40030   C3D1BC      JP &BCD1
19    40033   669C        40038          *
20    40035   C3939C      JP 40083       *
21    40038   46          'F'
22    40039   49          'I'
23    40040   58          'X'
24    40041   4F          'O'
25    40042   46          'F'
26    40043   C6          'F'+&80
27    40044   00          Fin du mot

28    40045   00          4 octets
29    40046   00          Reserves
30    40047   00          KERNEL
31    40048   00

32    40049   FE01        CP 1
33    40051   C0          RET NZ

34    40052   21AD9C      LD HL,40109    *
35    40055   73          LD (HL),E
36    40056   21B99C      LD HL,40121    *
```

```
37    40059   73          LD (HL),E
38    40060   215ABB      LD HL,&BB5A
39    40063   36C3        LD (HL),&C3
40    40065   23          INC HL
41    40066   36AA        LD (HL),&AA    *
42    40068   23          INC HL
43    40069   369C        LD (HL),&9C    *
44    40071   2175BB      LD HL,&BB75
45    40074   36C3        LD (HL),&C3
46    40076   23          INC HL
47    40077   36B6        LD (HL),&B6    *
48    40079   23          INC HL
49    40080   369C        LD (HL),&9C    *
50    40082   C9          RET


51    40083   215ABB      LD HL,&BB5A
52    40086   36CF        LD (HL),&CF
53    40088   23          INC HL
54    40089   3600        LD (HL),&00    #
55    40091   23          INC HL
56    40092   3694        LD (HL),&94    #
57    40094   2175BB      LD HL,&BB75
58    40097   36CF        LD (HL),&CF
59    40099   23          INC HL
60    40100   3674        LD (HL),&74    #
61    40102   23          INC HL
62    40103   3691        LD (HL),&91    #
63    40105   C9          RET


64    40106   F5          PUSH AF
65    40107   E5          PUSH HL
66    40108   3E00        LD A,0
67    40110   CDB4BB      CALL &BBB4
68    40113   E1          POP HL
69    40114   F1          POP AF
70    40115   CF0094      RST 8          #
71    40118   F5          PUSH AF
72    40119   E5          PUSH HL
73    40120   3E00        LD A,0
74    40122   CDB4BB      CALL &BBB4
75    40125   E1          POP HL
```

```
76    40126  F1          POP AF
77    40127  CF7491      RST 8          #
```

Les lignes 1 à 15 concernent l'intégration de la commande FIXON. Une explication théorique détaillée du mécanisme de l'intégration proprement dite n'est peut-être pas ce qu'il y a de plus intéressant, dans la mesure où l'on peut se contenter de le reproduire pour en réaliser d'autres.

Le registre BC doit tout d'abord être chargé avec l'adresse de ce que l'on appelle une table d'instruction, composée de :

1. L'adresse d'une zone de 4 octets réservés au système (40020 dans notre exemple). Attention, ces 4 octets doivent impérativement être implantés entre &4000 et &BFFF.
2. Un saut à l'adresse où est implantée la routine que l'on souhaite intégrer.
3. Les codes de chacune des lettres du nom de la routine, la valeur &80 devant être ajoutée à la dernière lettre et la fin du mot devant être signalée par un 0.

Le registre HL doit ensuite être chargé avec l'adresse des 4 octets réservés, puis l'intégration est installée par l'appel du vecteur &BCD1.

Un seul CALL 40000 sera donc nécessaire et suffisant avant que l'on puisse appeler la routine par son nom.

L'intégration de la commande |FIXOFF est réalisée d'une manière identique par les lignes 16 à 31.

La routine |FIXON commence en 40049 par charger le paramètre NF (contenu dans DE au moment de l'appel) dans les adresses 40109 et 40121. Nous verrons pourquoi plus loin.

Les lignes 38 à 50 effectuent ensuite le détournement des vecteurs &BB5A (sortie de caractère) et &BB75 (positionnement du curseur), respectivement vers les adresses 40106 et 40118. Le reste coule de source : avant les RESTART correspondants (lignes 40115 ou 40125), la fenêtre souhaitée est fixée par l'appel du vecteur &BBB4. Vous pouvez constater qu'avant cet appel le registre A est chargé avec la valeur 0... mais rappelez-vous qu'en réalité les emplacements mémoire 40109 et 40121 ont été chargés avec le numéro de fenêtre dès le début du programme (voir plus haut). Il s'agit donc en quelque sorte d'un programme qui s'automodifie.

Remarque

Il est à noter que l'instruction CLS est également fixée, bien que le vecteur spécifique d'effacement de la fenêtre courante (&BB6C) n'ait pas été détourné. Cela s'explique par le fait que les programmeurs du 464 ont choisi, pour le traitement de l'instruction BASIC CLS, l'option du code de contrôle d'effacement (12 ou &0C), qui est bien sûr envoyé via le vecteur &BB5A. Cela a été modifié sur le 664 et le 6128, pour lesquels il est certainement nécessaire de détourner &BB6C.

L'instruction |FIXOFF renvoie aux lignes 51 à 63, qui remettent les deux vecteurs concernés en l'état initial.

Voici le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 9
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 01499C21549CC3D1BC4E9CC371,665
170 DATA 9C4649584FCE00000000000161,302
180 DATA 9C216D9CC3D1BC669CC3939C46,750
190 DATA 49584F46C60000000000FE01C0,3BB
200 DATA 21AD9C7321B99C73215ABB36C3,5F5
210 DATA 2336AA23369C2175BB36C32336,49B
220 DATA B623369CC9215ABB36CF233600,508
230 DATA 2336942175BB36CF2336742336,469
240 DATA 91C9F5E53E00CDB4BBE1F1CF00,84F
250 DATA 94F5E53E00CDB4BBE1F1CF7491,88E
254 '
255 '
```

```
256 '            DEMONSTRATION
257 '
260 CALL 40000:CALL 40024
270 WINDOW#7,15,25,10,15:PAPER#7,3
280 |FIXON,7
290 CLS:LOCATE 4,3:PRINT"ESSAI"
300 |FIXOFF
```

■ ADAPTATION AU 664 ET AU 6128

Nous avons déjà vu que l'aspect en mémoire du vecteur &BB5A est différent, sur ces deux machines, de celui du 464. Il en est de même pour le vecteur &BB75 qui se présente comme suit :

Pour le 664

PEEK(&BB75) = &CF
PEEK(&BB76) = &6C
PEEK(&BB77) = &91

Dans le programme de chargement BASIC, les lignes suivantes devront par conséquent être modifiées :

```
220 DATA B623369CC9215ABB36CF2336FA,602
230 DATA 2336932175BB36CF2366C2336,460
240 DATA 91C9F5E53E00CDB4BBE1F1CFFA,949
250 DATA 93F5E53E00CDB4BBE1F1CF6C91,885

290 CLS#7:LOCATE 4,3:PRINT"ESSAI"
```

Pour le 6128

PEEK(&BB75) = &CF
PEEK(&BB76) = &70
PEEK(&BB77) = &91

Les lignes suivantes devront être modifiées :

```
220 DATA B623369CC9215ABB36CF2336FE,606
230 DATA 2336932175BB36CF2336702336,464
```

```
240 DATA 91C9F5E53E00CDB4BBE1F1CFFE,94D
250 DATA 93F5E53E00CDB4BBE1F1CF7091,889

290 CLS#7:LOCATE 4,3:PRINT"ESSAI"
```

8 COMMANDE BIP

Ce programme est destiné à remplacer avantageusement le PRINT CHR$(7) qu'offre le CPC : il permet l'envoi d'un nombre N de coups de cloche, chacun d'entre eux ayant une durée T (exprimée en dixièmes de seconde), séparés par un "bruit blanc" d'une durée identique, avec indication optionnelle d'une période P.

La routine est intégrée sous l'appellation |BIP, et son format d'appel est donc le suivant :

|BIP, N, T, P.

- Si le paramètre **P** est omis, la période par défaut est 239 (&EF), soit le *do* de la première octave.

Voici le listing assembleur :

```
1     40000   01499C     LD BC,40009    *
2     40003   21529C     LD HL,40018    *
3     40006   CDD1BC     JP &BCD1

4     40009   4E9C       40014          *
5     40011   C3569C     JP 40022       *
6     40014   42         'B'
7     40015   49         'I'
8     40016   D0         'P'+&80
9     40017   00         Fin du mot

10    40018   00         4 octets
11    40019   00         reserves
12    40020   00         Kernel
13    40021   00

14    40022   FE02       CP 2
15    40024   200C       JRNZ 40038
16    40026   DD2B       DEC IX
17    40028   DD2B       DEC IX
18    40030   21EF00     LD HL,239
```

```
19    40033   22AB9C      LD (40107),HL *
20    40036   1806        JR 40044
21    40038   FE03        CP 3
22    40040   C0          RET NZ
23    40041   EB          EX HL,DE
24    40042   18F5        JR 40033

25    40044   DD6603      LD H,(IX+3)
26    40047   DD6E02      LD L,(IX+2)
27    40050   E5          PUSH HL
28    40051   D1          POP DE
29    40052   0609        LD B,9
30    40054   19          ADD HL,DE
31    40055   10FD        DJNZ 40054
32    40057   22AF9C      LD (40111),HL *
33    40060   DD6605      LD H,(IX+5)
34    40063   DD6E04      LD L,(IC+4)
35    40066   E5          PUSH HL
36    40067   21A89C      LD HL,40104   *
37    40070   CDAABC      CALL &BCAA
38    40073   30FB        JRNC 40070
39    40075   2AAB9C      LD HL,(40107) *
40    40078   E5          PUSH HL
41    40079   210000      LD HL,0
42    40082   22AB9C      LD (40107),HL *
43    40085   21A89C      LD HL,40104   *
44    40088   CDAABC      CALL &BCAA
45    40091   30FB        JRNC 40088
46    40093   E1          POP HL
47    40094   22AB9C      LD (40107),HL *
48    40097   E1          POP HL
49    40098   2B          DEC HL
50    40099   7C          LD A,H
51    40100   B5          OR L
52    40101   20DB        JRNZ 40066
53    40103   C9          RET

              Queue sonore

54    40104   01          Canal
55    40105   00          Env. volume
56    40106   00          Env. ton
57    40107   0000        Periode
```

```
58    40109   00          Bruitage
59    40110   0C          Volume initial
60    40111   0000        Duree en 100emes
```

Le programme d'intégration (qui ne doit être appelé qu'une seule fois) est constitué par les lignes 1 à 13. Le procédé est le même que celui qui a été étudié au chapitre précédent.

La routine elle-même débute à l'adresse 40022 (ligne 14) par le test du nombre d'arguments transmis. Voyons tout d'abord ce qui se passe s'il y en a trois.

La période contenue dans DE est chargée dans le registre HL (ligne 23), puis un saut relatif à l'adresse 40033 est effectué. La valeur contenue dans HL est alors chargée dans l'emplacement mémoire 40107 de la queue sonore implantée en fin de programme. La période ayant été installée, le programme saute vers la partie commune (adresse 40044, ligne 25) qui a été conçue dans l'hypothèse que trois paramètres seulement seraient transmis.

Voyons maintenant le cas où il n'y en a que deux.

Le registre IX subit tout d'abord deux décrémentations successives et se retrouve donc pointé exactement comme si trois paramètres avaient été transmis (revoir éventuellement à ce sujet l'explicatif du Programme 6). La période par défaut (239) est ensuite chargée dans HL, puis installée à la place adéquate de la queue sonore. Le programme passe enfin à la partie commune.

Les lignes 25 à 32 traitent tout d'abord le paramètre T. Nous avons dit en effet qu'il devait être exprimé en dixièmes de seconde (ce qui nous a paru une unité plus simple à utiliser que les centièmes), alors que le temps, dans une queue sonore, doit toujours être exprimé en centièmes. Pour obtenir l'équivalence, il est donc nécessaire de multiplier le paramètre T par 10 avant de l'installer dans la queue. Ce paramètre est donc chargé dans HL, puis dans DE (lignes 27 et 28), et la boucle 30-31 additionne neuf fois de suite ces deux registres, B servant de compteur. Si l'on tient compte du fait que HL contenait déjà la valeur T, le résultat qui se retrouve dans ce registre après la boucle est bien $T+(9*T)=10*T$.

L'émission des sons est assurée par la boucle constituée des lignes 36 à 47. Elle est contrôlée par HL, qui sert de compteur et est chargé initialement avec le paramètre N (lignes 33 et 34). Le contenu de ce registre est préservé à chaque passe (ligne 35), récupéré en ligne 48, décrémenté en ligne 49 et testé par les lignes 50 et 51. Si HL atteint

0, la main est rendue au BASIC ; sinon la boucle reprend en ligne 35.

La boucle elle-même peut être séparée en deux parties essentielles. La première sort un son de période P (ou 239 selon le cas), et la seconde sort un bruit blanc d'une durée égale.

Comme il a déjà été vu dans le Programme 3, l'exécution d'une queue sonore et obtenue par l'appel du vecteur &BCAA, après pointage de HL sur le début de cette queue sonore. Tout cela est réalisé par les lignes 36 et 37, mais cette fois la nouveauté réside dans le saut relatif de la ligne 38, qui a lieu au retour de l'appel si aucun report n'a été généré.

Il peut arriver, en effet, que la file d'attente de traitement des queues sonores soit complètement remplie et que l'installation d'une queue supplémentaire par le programme soit, de ce fait, impossible. (Il ne faut pas oublier que le programme, après avoir déposé une queue, n'attend pas qu'elle soit traitée pour continuer son travail. Pour ce qui le concerne, il peut donc tout aussi bien présenter le dixième ou le vingtième son alors même que le premier est encore en cours de traitement. Or la file d'attente n'a pas une contenance illimitée...) Les programmeurs ont naturellement prévu cette éventualité de saturation et la parade est la suivante : si le son a pu être installé, le sémaphore de report est vrai. Dans le cas contraire, le sémaphore de report est faux. Il suffit donc de le tester au retour de &BCAA et, le cas échéant, de faire boucler le programme jusqu'à ce qu'une place se libère.

Notre queue sonore étant à présent tranquillement installée dans la file, il va falloir réaliser une deuxième fois la même opération mais dans le but, cette fois, de produire un silence (ou plus exactement un bruit blanc, puisque nous allons réutiliser la même queue sonore en nous contentant de remplacer provisoirement la fréquence précédente par 0).

Les lignes 39 et 40 commencent par sauvegarder la fréquence initiale en la rangeant sur la pile. Les lignes 41 et 42 chargent ensuite la valeur 0 dans l'emplacement mémoire correspondant à la fréquence, et ce bruit blanc est ensuite rangé dans la file de la même manière que précédemment (lignes 43, 44 et 45). La fréquence est enfin récupérée et remise en place, en prévision de la passe suivante.

La boucle est donc bouclée, et l'ensemble du processus va se reproduire N fois.

Voici le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 8
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 01499C21529CC3D1BC4E9CC356,648
170 DATA 9C4249D00000000000FE02200C,323
180 DATA DD2BDD2B21EF0022AB9C1806FE,5A5
190 DATA 03C0EB18F5DD6603DD6E02E5D1,704
200 DATA 06091910FD22AF9CDD6605DD6E,535
210 DATA 04E521A89CCDAABC30FB2AAB9C,71D
220 DATA E521000022AB9C21A89CCDAABC,607
230 DATA 30FBE122AB9CE12B7CB520DBC9,776
240 DATA 0100000000000C0000,D
250 '
260 '
270 '            DEMONSTRATION
280 '
290 CALL 40000
300 PRINT:PRINT:PRINT"PRESSEZ UNE TOUCHE
"
310 REP$=INKEY$:IF REP$="" THEN 310
320 |BIP,4,2,300
330 REP$=INKEY$:IF REP$="" THEN 330
340 FOR I=250 TO 270:|BIP,1,1,I:NEXT
350 REP$=INKEY$:IF REP$="" THEN 350
360 POKE 40080,&C8:|BIP,10,1,300
370 POKE 40080,0
```

Cette routine fonctionne indifféremment sur les trois machines.

9 ALARME

Si vous avez acheté ce livre, il y a de fortes chances pour que vous fassiez partie de la grande confrérie des drogués de la programmation. Sinon, il est probable que cela soit en bonne voie...

Qui d'entre vous, perdu dans son *trip* informatique, n'a pas un jour manqué un rendez-vous, oublié de manger ou, pire encore, laissé passer l'heure du ciné-club ?

Pour ce qui me concerne, cela m'est arrivé un nombre suffisant de fois pour que je me décide à mettre au point la routine suivante, qui a pour fonction d'émettre une sonnerie après un délai précis.

Ce programme utilise la technique des interruptions et ne peut donc être perturbé par aucune instruction BASIC ou manipulation du clavier (ce qui n'est pas le cas avec l'instruction EVERY, qui se bloque par exemple pendant les saisies de variables par INPUT). Il peut d'autre part être lancé en mode direct, et vous pourrez travailler sur la machine tout à fait normalement, exactement comme si vous aviez mis un réveil près de vous. Il va de soi que l'émission de la sonnerie est sans danger pour le travail éventuellement en cours.

Le format d'appel de cette routine est le suivant :

CALL 40000, T

- Le paramètre **T** représente donc le temps, exprimé en minutes, que vous souhaitez voir s'écouler entre le lancement de la routine et l'émission de la sonnerie.

Quelques mots sur les interruptions seront probablement les bienvenus pour un certain nombre d'entre vous. Précisons tout de suite que nous allons simplifier au maximum ; ce sujet est en effet fort complexe, et une étude détaillée dépasserait très largement le cadre du présent ouvrage.

Les interruptions sont des phénomènes qui obligent le système interne à effectuer des opérations précises à des moments précis. L'exemple le plus évident est celui du *scanning* de clavier (*to scan* signifie scruter en anglais). Si vous avez l'impression, lorsque vous pressez une touche, que l'ordinateur réagit immédiatement, c'est tout simplement parce qu'il se produit, tous les cinquantièmes de seconde

et quel que soit le travail en cours par ailleurs, une interruption qui branche le système sur une routine d'analyse du clavier. D'autres interruptions se produisent par exemple pour gérer les circuits vidéo, en synchronisation avec le balayage de l'écran par le rayon (qui se reproduit lui aussi tous les cinquantièmes de seconde).

Précisons encore, pour information, qu'il existe différents types d'interruptions, certaines ayant lieu tous les 1/300 de seconde, d'autres tous les 1/100 de seconde, d'autres enfin tous les 1/50 de seconde.

Tout cela est bien sûr l'affaire personnelle du système interne et n'aurait guère pour nous qu'un intérêt tout théorique s'il n'était possible de programmer des interruptions de notre propre cru. Il faut pour cela transmettre un certain nombre d'informations qui vont constituer ce que l'on appelle un *event-bloc* (littéralement : ''bloc d'événement'') dont la constitution dépend du type d'interruption que l'on souhaite mettre en place. Pour ce qui nous concerne, nous ne nous intéresserons qu'aux interruptions de type *frame flyback*, synchronisées avec le balayage de l'écran et ayant lieu tous les 1/50 de seconde.

Dans ce cas précis, l'*event-bloc* doit comprendre 9 octets. Nous nous contenterons de mettre à 0 les cinq premiers, en les laissant aux bons soins du système qui les utilisera pour gérer l'interruption.

- Le neuvième octet ne sert qu'à indiquer dans quelle zone mémoire (RAM, ROM interne, ROM externe, etc.) est implantée la routine qui devra être appelée à chaque interruption. Si cette routine est en RAM centrale (ce qui est le cas pour notre alarme), cet octet doit être mis à 0.
- Les septième et huitième octets indiquent l'adresse de la routine.
- Le sixième octet, enfin, indique la ''classe'' de l'événement. Chaque bit de cet octet a un rôle bien précis :

Bit 0

Il doit être à 1 si la routine est implantée en RAM centrale ou en ROM inférieure ; il doit être à 0 si la routine est implantée en ROM supérieure.

Bits 1 à 4

Ils déterminent le niveau de priorité de l'*event-bloc* (n'oublions pas qu'il y a pas mal d'interruptions et que le système doit les traiter les unes après les autres, par ordre d'importance).

Bit 5

Il doit toujours être mis à 0.

Bit 6

Il doit être à 1 si l'événement est de type *express*, et à 0 si l'événement est de type *normal* (cela recouvre une autre hiérarchie de priorité).

Bit 7

Il doit être à 1 si l'événement est de type *asynchrone*, et à 0 si l'événement est de type *synchrone*. Les événements asynchrones sont traités immédiatement et n'ont pas de file d'attente. Les événements synchrones ne sont traités que lorsque le système a le temps.

Tout cela sera sans doute moins nébuleux lorsque vous vous serez amusé un peu à manipuler les interruptions.

Pour en revenir à notre routine, l'octet de classe est &81, soit en binaire :

Bit 0=1
Bit 1=0
Bit 2=0
Bit 3=0
Bit 4=0
Bit 5=0
Bit 6=0
Bit 7=1

Voici le listing assembleur du programme :

```
1     40000  FE01       CP 1
2     40002  C0         RET NZ

3     40003  EB         EX HL,DE
4     40004  22889C     LD (40072),HL *
5     40007  21B80B     LD HL,3000
6     40010  22869C     LD (40070),HL *
7     40013  218A9C     LD HL,40074   *
8     40016  CDDABC     CALL &BCDA
9     40019  C9         RET
```

```
10    40020   2A869C      LD HL,(40070) *
11    40023   2B          DEC HL
12    40024   7C          LD A,H
13    40025   B5          OR L
14    40026   22869C      LD (40070),HL *
15    40029   C0          RETNZ
16    40030   21B80B      LD HL,3000
17    40033   22869C      LD (40070),HL *
18    40036   2A889C      LD HL,(40072) *
19    40039   2B          DEC HL
20    40040   7C          LD A,H
21    40041   B5          OR L
22    40042   22889C      LD (40072),HL *
23    40045   C0          RETNZ
24    40046   0664        LD B,100
25    40048   210010      LD HL,16
26    40051   3E07        LD A,7
27    40053   CD5ABB      CALL &BB5A
28    40056   2B          DEC HL
29    40057   7C          LD A,H
30    40058   B5          OR L
31    40059   20FB        JRNZ 40056
32    40061   10F1        DJNZ 40048
33    40063   218A9C      LD HL,40074   *
34    40066   CDDDBC      CALL &BCDD
35    40069   C9          RET

36    40070   0000        Placard 1
37    40072   0000        Placard 2


                Event-bloc

38    40074   00
39    40075   00
40    40076   00
41    40077   00
42    40078   00
43    40079   81          Classe
44    40080   549C        Adr. routine  *
45    40082   00
```

Les lignes 1 à 9 préparent et initialisent la routine d'interruption, cette dernière étant constituée par les lignes 10 à 35.

Nous avons dit que l'interruption aurait lieu tous les 1/50 de seconde ; le paramètre T étant censé représenter des minutes, deux compteurs vont être nécessaires :

- Le premier sera initialisé avec la valeur 3000. Il n'est pas bien difficile d'en déduire qu'il n'atteindra la valeur 0 qu'une fois par minute (3 000 fois 1/50 de seconde = 60 secondes).
- Le second sera naturellement initialisé avec le paramètre T.

Avant toute mise en service de l'interruption, ces deux valeurs sont rangées respectivement dans les placards 2 et 1 (lignes 3 à 6). Ensuite seulement notre interruption est activée de la manière la plus simple du monde : il suffit de charger HL avec l'adresse de l'*event-bloc* et d'appeler le vecteur &BCDA avant de rendre la main au BASIC. A partir de cet instant et sans qu'aucune autre intervention soit nécessaire, le système va se brancher sur la routine d'adresse 40020 tous les 1/50 de seconde.

Voyons comment fonctionne cette routine :

Le registre HL est chargé en ligne 10 avec la valeur contenue dans le placard 2 ; il est ensuite décrémenté et testé (lignes 12 et 13). Deux cas peuvent alors se présenter :

- Soit il contient encore une valeur supérieure à 0 ; dans ce cas, cette valeur est rangée dans le placard 2 (ligne 14) et la routine se termine là (ligne 15).
- Soit il vaut 0 ; dans ce cas, le placard 2 est d'abord réinitialisé (lignes 16 et 17), puis une opération similaire est réalisée avec le contenu du placard 1 (lignes 18 à 23 : chargement de HL, décrémentation, puis test). Là encore, deux possibilités : si HL est différent de 0, la routine s'arrête ; sinon elle se poursuit en ligne 24.

Pour résumer, le premier compteur forme une sorte de verrou n'autorisant qu'une décrémentation par minute du deuxième compteur. Lorsque ce dernier arrive à 0, le délai prévu est écoulé.

La sonnerie est constituée de 100 coups de cloche obtenus grâce au code de contrôle 7, par l'intermédiaire du vecteur &BB5A. Une temporisation (16 décrémentations de HL) est intercalée entre chacun d'entre eux, de manière que la sonnerie ne se transforme pas en sirène... Nous avons donc deux boucles imbriquées : la boucle

secondaire de temporisation (lignes 28 à 31), et la boucle principale d'émission des coups de cloche (lignes 25 à 32).

Une fois tout cela terminé, il ne reste plus qu'à détruire l'interruption, qui ne sert plus à rien. L'opération n'est pas bien difficile, puisqu'il suffit de charger HL avec l'adresse de l'*event-bloc* et d'appeler le vecteur &BCDD.

Voici enfin le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 6
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE01C0EB22889C21B80B22869C,618
170 DATA 218A9CCDDABCC92A869C2B7CB5,71B
180 DATA 22869CC021B80B22869C2A889C,57A
190 DATA 2B7CB522889CC006642100103E,43B
200 DATA 07CD5ABB2B7CB520FB10F1218A,60C
210 DATA 9CCDDDBCC90000000000000000,3CB
220 DATA 0081549C,171
230 '
240 '
250 '            DEMONSTRATION
260 '
270 CALL 40000,1
280 PRINT"QUOI QUE VOUS FASSIEZ ENTRE-TE
MPS,CA VA SONNER DANS UNE MINUTE..."
```

Ce programme fonctionne sans modification sur les trois machines.

10 CHRONOMÈTRE EN FENÊTRE 7

Le système des interruptions est naturellement idéal pour n'importe quel programme ayant à voir de près ou de loin avec le comptage du temps. Cette application est loin d'être la seule, mais disons qu'elle est la plus évidente.

La routine suivante était donc presque obligatoire, qui permet l'affichage en fenêtre 7 d'un chronomètre indiquant les heures, minutes et secondes (cette fenêtre devra bien sûr avoir été définie au préalable).

Quatre adresses peuvent en réalité être sollicitées :

CALL 40000, NC	Ce format déclenche le chronomètre. Le paramètre NC est obligatoire et doit indiquer le numéro de colonne gauche de la fenêtre 7.
CALL 40193	Ce format fige le chronomètre dans l'état, mais sans détruire l'*event-bloc*.
CALL 40197	Ce format déclenche à nouveau le chronomètre en partant des valeurs figées.
CALL 40203	Ce format détruit l'*event-bloc* et restaure certaines valeurs pour préparer le programme à un éventuel appel ultérieur.

Pour une bonne compréhension du programme, quelques points doivent être précisés :

☐ Le chronomètre se présentera sur l'écran de la manière suivante (chaque x représente un chiffre compris entre 0 et 6 ou entre 0 et 10, selon qu'il s'agit par exemple des secondes ou des dizaines de secondes) :

xhxxmxxs

Le chiffre précédant les heures devra par exemple être affiché dans la première colonne de la fenêtre 7. En BASIC, cela serait résolu par un bon vieux LOCATE et ne poserait donc aucun problème. En lan-

gage machine non plus, d'ailleurs, si l'on utilisait le vecteur &BB75 (positionnement du curseur). Mais il se trouve que nous avons écarté cette option pour en choisir une autre qui, si elle est un peu plus complexe, a l'avantage d'être beaucoup plus rapide. D'une manière générale, en effet, les routines utilisées en interruption ne doivent pas avoir une durée d'exécution trop longue si l'on veut éviter certains désagréments. Or la nôtre est relativement importante (plus de 200 octets) et comporte évidemment des procédures d'affichage qui sont de grosses consommatrices de temps. Il s'agit donc d'économiser partout où on le peut.

Pour en revenir au positionnement du curseur, il s'effectuera par le biais d'une variable système située en &B286 (&B727 pour le 664 et le 6128) et qui indique la position horizontale du curseur dans la fenêtre courante. Il est tout à fait possible de modifier le contenu de cet emplacement mémoire, et de "forcer" ainsi le curseur à une position donnée. Il faut néanmoins savoir que la position indiquée doit être relative à une ligne d'écran normale, en admettant que cette ligne aille de la colonne 0 à la colonne 39 (si par exemple la fenêtre 7 est définie comme commençant à la colonne 16 de l'écran, le positionnement du curseur à la colonne 1 de la fenêtre se fera en chargeant &B286 avec la valeur 15, le positionnement à la colonne 2 en chargeant &B286 avec 16, le positionnement en colonne 3 en chargeant &B286 avec 17, etc.).

☐ Cinq compteurs vont être nécessaires (secondes, dizaines de secondes, minutes, dizaines de minutes, heures). Ces compteurs seront conçus de telle manière qu'ils puissent avoir une double fonction : servir de compteurs et indiquer quels sont les caractères qui doivent être affichés.

Les codes ASCII &30 à &3A représentent les caractères suivants :

&30 : "0"
&31 : "1"
&32 : "2"
&33 : "3"
&34 : "4"
&35 : "5"
&36 : "6"
&37 : "7"
&38 : "8"

&39 : ''9''

&3A : '':''

Prenons l'exemple des secondes : si le compteur correspondant est initialisé avec &30, il suffira de vérifier, à chaque incrémentation, si la valeur &3A est atteinte. Si ce n'est pas le cas, le caractère correspondant au contenu du compteur devra être affiché (ce sera donc successivement 1, 2, 3, etc.). Si par contre la valeur &3A est atteinte, le compteur des secondes devra être remis à &30 et le compteur des dizaines de secondes incrémenté de 1.

Le test sur les dizaines de secondes se fera sur la valeur &36 : si celle-ci est atteinte, cela représente soixante secondes. Le compteur des dizaines de secondes devra donc être remis à &30, et le compteur des minutes devra être incrémenté de 1.

Les incrémentations et les tests se répercuteront éventuellement ainsi jusqu'au compteur des heures.

Voici le listing assembleur :

```
1     40000  FE01       CP 1
2     40002  C0         RET NZ

3     40003  3E07       LD A,7
4     40005  CDB4BB     CALL &BBB4
5     40008  1D         DEC E
6     40009  7B         LD A,E
7     40010  32869C     LD (&B286),A   #
8     40013  32C49C     LD (40132),A   *
9     40016  3C         INC A
10    40017  3C         INC A
11    40018  32D09C     LD (40144),A   *
12    40021  3C         INC A
13    40022  32DC9C     LD (40156),A   *
14    40025  3C         INC A
15    40026  3C         INC A
16    40027  32E89C     LD (40168),A   *
17    40030  3C         INC A
18    40031  32F49C     LD (40180),A   *
19    40034  21709C     LD HL,40048    *
20    40037  0608       LD B,8
21    40039  7E         LD A,(HL)
22    40040  CD5ABB     CALL &BB5A
23    40043  23         INC HL
```

```
24    40044   10F9        DJNZ 40039
25    40046   1808        JR 40056

26    40048   30          '0'
27    40049   68          'h'
28    40050   30          '0'
29    40051   30          '0'
30    40052   6d          'm'
31    40053   30          '0'
32    40054   30          '0'
33    40055   73          's'

34    40056   AF          XOR A
35    40057   CDB4BB      CALL &BBB4
36    40060   21249D      LD HL,40228   *
37    40063   CDDABC      CALL &BCDA
38    40066   C9          RET


39    40067   21239D      LD HL,40227   *
40    40070   35          DEC (HL)
41    40071   C0          RET NZ
42    40072   3632        LD (HL),50
43    40074   3E07        LD A,7
44    40076   CDB4BB      CALL &BBB4
45    40079   21229D      LD HL,40226   *
46    40082   34          INC (HL)
47    40083   3E3A        LD A,&3A
48    40085   BE          CP (HL)
49    40086   C2F39C      JP NZ 40179   *
50    40089   3630        LD (HL),&30
51    40091   2B          DEC HL
52    40092   3E36        LD A,&36
53    40094   34          INC (HL)
54    40095   BE          CP (HL)
55    40096   C2E79C      JP NZ 40167   *
56    40099   3630        LD (HL),&30
57    40101   2B          DEC HL
58    40102   3E3A        LD A,&3A
59    40104   34          INC (HL)
60    40105   BE          CP (HL)
61    40106   C2DB9C      JP NZ 40155   *
62    40109   3630        LD (HL),&30
```

```
63   40111  2B          DEC HL
64   40112  3E36        LD A,&36
65   40114  34          INC (HL)
66   40115  BE          CP (HL)
67   40116  C2CF9C      JP NZ 40143    *
68   40119  3630        LD (HL),&30
69   40121  2B          DEC HL
70   40122  3E3A        LD A,&3A
71   40124  34          INC (HL)
72   40125  BE          CP (HL)
73   40126  C2C39C      JP NZ 40131    *
74   40129  3630        LD (HL),&30

75   40131  3E00        LD A,0
76   40133  3286B2      LD (&B286),A   #
77   40136  7E          LD A,(HL)
78   40137  CD5DBB      CALL &BB5D
79   40140  211F9D      LD HL,40223    *
80   40143  3E00        LD A,0
81   40145  3286B2      LD (&B286),A   #
82   40148  7E          LD A,(HL)
83   40149  CD5DBB      CALL &BB5D
84   40152  21209D      LD HL,40224    *
85   40155  3E00        LD A,0
86   40157  3286B2      LD (&B286),A   #
87   40160  7E          LD A,(HL)
88   40161  CD5DBB      CALL &BB5D
89   40164  21219D      LD HL,40225    *
90   40167  3E00        LD A,0
91   40169  3286B2      LD (&B286),A   #
92   40172  7E          LD A,(HL)
93   40173  CD5DBB      CALL &BB5D
94   40176  21229D      LD HL,40226    *
95   40179  3E00        LD A,0
96   40181  3286B2      LD (&B286),A   #
97   40184  7E          LD A,(HL)
98   40185  CD5DBB      CALL &BB5D
99   40188  AF          XOR A
100  40189  CDB4BB      CALL &BBB4
101  40192  C9          RET

102  40193  3EC9        LD A,&C9
103  40195  1802        JR 40199
```

```
104   40197   3E21        LD A,&21
105   40199   32839C      LD (40067),A   *
106   40202   C9          RET

107   40203   21249D      LD HL,40228    *
108   40206   CDDDBC      CALL &BCDD
109   40209   211E9D      LD HL,40222    *
110   40212   0605        LD B,5
111   40214   3630        LD (HL),&30
112   40216   23          INC HL
113   40217   10FB        DJNZ 40214
114   40219   3632        LD (HL),&32
115   40221   C9          RET


116   40222   30          Heures
117   40223   30          Dizaines min.
118   40224   30          Minutes
119   40225   30          Dizaines sec.
120   40226   30          Secondes
121   40227   32          Compteur


                  Event-bloc

122   40228   00
123   40229   00
124   40230   00
125   40231   00
126   40232   00
127   40233   81          Classe
128   40234   839C        Adresse routine
129
      40236   00
```

Avant toute mise en route de l'interruption, quelques préparatifs sont effectués par les lignes 3 à 35 :

1. La fenêtre 7 est installée comme fenêtre courante (lignes 3 et 4).
2. Le paramètre NC, après avoir été décrémenté, est chargé dans A (lignes 5 et 6), puis dans l'emplacement mémoire &B286. Rap-

pelons que cela a pour effet de positionner le curseur à la première colonne de la fenêtre 7. Le but de l'opération est de préparer la sortie de chaîne des lignes 20 à 24 ; nous y reviendrons.

3. Les lignes 8 à 18 chargent les emplacements mémoire 40132, 40144, 40156, 40168 et 40180 respectivement avec NC−1, NC+1, NC+2, NC+4 et NC+5. En se reportant aux lignes 75, 80, 85, 90 et 95, on peut constater que ces emplacements mémoire doivent contenir les valeurs à charger dans &B286 avant les affichages des heures, dizaines de minutes, minutes, dizaines de secondes et secondes. Cela prépare donc les positionnements de curseur en colonnes 1, 3, 4, 6 et 7 de la fenêtre 7.

4. La chaîne de caractères écrite des adresses 40048 à 40055 est affichée par la boucle allant des lignes 21 à 24, B servant de compteur (ce n'est qu'en prévision de cet affichage que la fenêtre 7 a été définie au début du programme comme fenêtre courante et que le curseur a été positionné en colonne 1 de cette fenêtre).

Les préparatifs étant terminés, le saut de la ligne 25 renvoie aux lignes 34 et 35, où la fenêtre 0 (c'est-à-dire l'écran normal) est réinstallée. L'interruption est ensuite mise en service (lignes 36 et 37) et le retour effectué.

La routine commence à la ligne 29 par charger HL avec le contenu de l'emplacement mémoire 40227. Ce registre est alors décrémenté et testé. La valeur initiale contenue dans 40227 étant &32 ou 50 en décimal (voir ligne 121), le 0 ne sera atteint qu'une fois par seconde (50 fois 1/50 = 1 seconde). Lorsque cela se produit, le passage en ligne 42 se fait : le compteur est restauré (ligne 42) et la fenêtre 7 définie comme la fenêtre courante. Le registre HL est ensuite pointé sur la fin de la zone mémoire contenant les compteurs (c'est-à-dire sur le compteur des secondes ; voir lignes 116 à 120).

Les procédures d'incrémentation et de test des compteurs peuvent alors commencer :

Lignes 46 à 50

Incrémentation du compteur des secondes et comparaison avec la valeur &3A.

- Si cette valeur n'est pas atteinte, le saut est effectué à l'adresse 40179, ligne 95 : là, le curseur est positionné en colonne 7 de la

fenêtre 7 par les lignes 95 et 96 (n'oublions pas que l'emplacement mémoire 40180 a été chargé de manière adéquate lors des préparations). Le code ASCII contenu dans le compteur des secondes est ensuite affiché (lignes 97 et 98). Pour finir, la fenêtre 0 est réinstallée et le retour effectué.

- Si la valeur &3A est atteinte, le compteur des secondes est restauré (ligne 50) et le registre HL est pointé sur le compteur des dizaines de secondes (ligne 51).

Lignes 52 à 56

Incrémentation du compteur des dizaines de secondes et comparaison avec la valeur &36.

- Si cette valeur n'est pas atteinte, saut à l'adresse 40167 : positionnement du curseur en colonne 6, affichage des dizaines de secondes, positionnement du curseur en colonne 7, affichage des secondes, et réinstallation de la fenêtre 0.
- Si la valeur &36 est atteinte, le compteur des dizaines de secondes est restauré, et le registre HL pointé sur le compteur des minutes.

Lignes 58 à 62

Incrémentation du compteur des minutes et comparaison avec la valeur &3A.

Lignes 64 à 68

Incrémentation du compteur des dizaines de minutes et comparaison avec la valeur &36.

Lignes 70 à 74

Incrémentation du compteur des heures et comparaison avec la valeur &3A.

Nous avons dit que le chronomètre pouvait être figé par l'appel de l'adresse 40193. Le principe est extrêmement simple : le registre A est chargé avec &C9 (ligne 103), puis le saut de la ligne 103 renvoie à l'adresse 40199, où la valeur contenue dans A est chargée dans l'emplacement mémoire 40067, c'est-à-dire le premier de la routine.

A partir de cet instant, donc, la première instruction de la routine est l'instruction de retour &C9. Il est par conséquent bien évident qu'aucune opération n'aura lieu et que le chronomètre sera laissé en l'état.

Le redémarrage se fait tout naturellement en remettant la valeur initiale &21 dans l'emplacement mémoire 40067 (lignes 104 et 105).

La mise hors service de l'interruption est obtenue par l'appel de l'adresse 40203 : l'*event-bloc* est détruit par les lignes 107 et 108. Les cinq compteurs sont ensuite réinitialisés à &30 par la boucle 111 à 113 ; à la fin de cette boucle, HL se retrouve pointé sur le sixième compteur qui est réinitialisé à &32. Le chronomètre est ainsi prêt pour un appel ultérieur.

Voici maintenant le programme de chargement BASIC.

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 18
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR E
N LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE01C03E07CDB4BB1D7B3286B2,642
170 DATA 32C49C3C3C32D09C3C32DC9C3C,5CA
180 DATA 3C32E89C3C32F49C21709C0608,52B
190 DATA 7ECD5ABB2310F9180830683030,4A4
200 DATA 6D303073AFCDB4BB21249DCDDA,6B4
210 DATA BCC921239D35C036323E07CDB4,589
220 DATA BB21229D343E3ABEC2F39C3630,5BC
230 DATA 2B3E3634BEC2E79C36302B3E3A,4DF
240 DATA 34BEC2DB9C36302B3E3634BEC2,5E4
250 DATA CF9C36302B3E3A34BEC2C39C36,5BD
260 DATA 303E003286B27ECD5DBB211F9D,518
270 DATA 3E003286B27ECD5DBB21209D3E,527
```

```
280 DATA 003286B27ECD5DBB21219D3E00,4EA
290 DATA 3286B27ECD5DBB21229D3E0032,51D
300 DATA 86B27ECD5DBBAFCDB4BBC93EC9,856
310 DATA 18023E2132839CC921249DCDDD,51F
320 DATA BC211E9D060536302310FB3632,39F
330 DATA C9303030303032000000000081,26C
340 DATA 839C,11F
350 '
360 '            DEMONSTRATION
370 '
380 MODE 1:LOCATE 10,7
390 PRINT"LE CHRONO FONCTIONNE"
400 LOCATE 15,9:PRINT"EN MODE 1"
410 WINDOW#7,16,23,12,12
420 PAPER#7,3
430 CALL 40000,16
440 FOR I=1 TO 12000:NEXT I
450 CALL 40203
460 '
470 MODE 0:LOCATE 4,7
480 PRINT"ET EN MODE 0"
490 WINDOW#7,6,13,12,12
500 CALL 40000,6
510 FOR I=1 TO 12000:NEXT I
520 CALL 40203
```

■ ADAPTATION AU 664 ET AU 6128

Sur ces deux machines, la variable système indiquant la position horizontale du curseur est rangée en &B727 au lieu de &B286. Six lignes devront être modifiées dans le programme de chargement BASIC :

```
160 DATA FE01C03E07CDB4BB1D7B3227B7,5E8

260 DATA 303E003227B77ECD5DBB211F9D,4BE
270 DATA 3E003227B77ECD5DBB21209D3E,4CD
280 DATA 003227B77ECD5DBB21219D3E00,490
290 DATA 3227B77ECD5DBB21229D3E0032,4C3
300 DATA 27B77ECD5DBBAFCDB4BBC93EC9,7FC
```

11 COLORATION DE ZONES (255 COULEURS)

Le programme que nous vous proposons maintenant est une simulation de l'instruction BASIC PAINT, avec en plus l'intéressante particularité d'offrir une palette de 255 pseudo-couleurs. Vous pourrez constater, en lançant le programme de chargement et démonstration dont le listing se trouve en fin de chapitre, que le résultat est tout à fait superbe.

Le format d'appel de la routine est le suivant :

CALL 40000, X, Y, C

- Les paramètres **X** et **Y** doivent indiquer les coordonnées d'un point quelconque situé à l'intérieur de la figure à colorer.
- Le paramètre **C** doit indiquer un numéro de couleur entre 1 et 255.

Quelques précautions d'emploi sont à signaler :

☐ Les coordonnées transmises ne doivent pas dépasser ou être inférieures aux coordonnées légales (de 0 à 639 dans le sens de la largeur, et de 0 à 399 dans le sens de la hauteur). Si une erreur était malgré tout commise, le programme ne sauterait en principe pas, mais rien ne se passerait sur l'écran pendant un laps de temps plus ou moins long, la routine effectuant alors de mystérieuses opérations en dehors de l'écran (en tout état de cause, vous pouvez toujours ajouter des contrôles adéquats si vous le souhaitez).

☐ Il est préférable que le point indiqué ne touche aucun des bords de la figure. Cette précaution n'est pas vitale mais elle évite des résultats inattendus.

☐ La figure doit être fermée. Méfiez-vous en particulier des cercles lorsqu'ils ont un grand rayon : les points ne sont pas toujours jointifs et la figure ne peut être considérée comme fermée. Là encore, aucun risque majeur ; vous risquez simplement de voir la couleur "s'étaler" un peu partout sur l'écran...

□ Il faut savoir également que les figures convexes dans le sens de la largeur seront remplies sans problème, mais que les figures convexes dans le sens de la hauteur devront généralement être traitées en plusieurs fois.

Une dernière précision, enfin : telle qu'elle est présentée, la routine fonctionne en mode 0 et en mode 1. Quatre octets devront être mis à 0 pour qu'elle puisse être utilisée en mode 2 : 40168, 40147, 40188 et 40163.

Voici le listing assembleur :

```
1     40000  FE03       CP 3
2     40002  C0         RET NZ

3     40003  21769C     LD HL,40054    *
4     40006  22E9BD     LD (&BEE9),HL
5     40009  23         INC HL
6     40010  73         LD (HL),E
7     40011  DD6603     LD H,(IX+3)
8     40014  DD6E02     LD L,(IX+2)
9     40017  DD5605     LD D,(IX+5)
10    40020  DD5E04     LD E,(IX+4)
11    40023  E5         PUSH HL
12    40024  D5         PUSH DE
13    40025  CDD29C     CALL 40146     *
14    40028  CDE69C     CALL 40166     *
15    40031  CD7B9C     CALL 40059     *
16    40034  D1         POP DE
17    40035  E1         POP HL
18    40036  2B         DEC HL
19    40037  2B         DEC HL
20    40038  CDD29C     CALL 40146     *
21    40041  CDE69C     CALL 40166     *
22    40044  CDA99C     CALL 40105     *
23    40047  21680C     LD HL,&0C68    #
24    40050  22E9BD     LD (&BDE9),HL
25    40053  C9         RET

26    40054  0600       LD B,0
27    40056  C3680C     JP &0C68       #

28    40059  C5         PUSH BC
```

```
29   40060  D5          PUSH DE
30   40061  23          INC HL
31   40062  23          INC HL
32   40063  E5          PUSH HL
33   40064  CDF0BB      CALL &BBF0
34   40067  FE00        CP 0
35   40069  2013        JRNZ 40090
36   40071  E1          POP HL
37   40072  E5          PUSH HL
38   40073  019001      LD BC,400
39   40076  ED42        SBC HL,BC
40   40078  E1          POP HL
41   40079  D1          POP DE
42   40080  C1          POP BC
43   40081  D0          RET NC
44   40082  CDD29C      CALL 40146     *
45   40085  CDE69C      CALL 40166     *
46   40088  18E1        JR 40059
47   40090  E1          POP HL
48   40091  2B          DEC HL
49   40092  2B          DEC HL
50   40093  EB          EX HL,DE
51   40094  E1          POP HL
52   40095  C1          POP BC
53   40096  23          INC HL
54   40097  23          INC HL
55   40098  CD0D9D      CALL 40205     *
56   40101  EB          EX HL,DE
57   40102  38D3        JRC 40059
58   40104  C9          RET

59   40105  C5          PUSH BC
60   40106  D5          PUSH DE
61   40107  2B          DEC HL
62   40108  2B          DEC HL
63   40109  E5          PUSH HL
64   40110  CDF0BB      CALL &BBF0
65   40113  FE00        CP 0
66   40115  200E        JRNZ 40131
67   40117  E1          POP HL
68   40118  D1          POP DE
69   40119  C1          POP BC
70   40120  CB7C        BIT 7.H
```

```
71   40122  C0          RET NZ
72   40123  CDD29C      CALL 40146     *
73   40126  CDE69C      CALL 40166     *
74   40129  18E6        JR 40105
75   40131  E1          POP HL
76   40132  23          INC HL
77   40133  23          INC HL
78   40134  EB          EX HL,DE
79   40135  E1          POP HL
80   40136  C1          POP BC
81   40137  23          INC HL
82   40138  23          INC HL
83   40139  CD0D9D      CALL 40205     *
84   40142  EB          EX HL,DE
85   40143  38D8        JRC 40105
86   40145  C9          RET

87   40146  E5          PUSH HL
88   40147  1B          DEC DE
89   40148  1B          DEC DE
90   40149  D5          PUSH DE
91   40150  CDF0BB      CALL &BBF0
92   40153  D1          POP DE
93   40154  E1          POP HL
94   40155  FE00        CP 0
95   40157  2004        JRNZ 40163
96   40159  CB7A        BIT 7,D
97   40161  28EF        JRZ 40146
98   40163  13          INC DE
99   40164  13          INC DE
100  40165  C9          RET

101  40166  D5          PUSH DE
102  40167  E5          PUSH HL
103  40168  13          INC DE
104  40169  13          INC DE
105  40170  D5          PUSH DE
106  40171  CDF0BB      CALL &BBF0
107  40174  D1          POP DE
108  40175  E1          POP HL
109  40176  FE00        CP 0
110  40178  2008        JRNZ 40188
111  40180  CB4A        BIT 1,D
```

```
112  40182  28EF      JRZ 40167
113  40184  CB7B      BIT 7,E
114  40186  28EB      JRZ 40167
115  40188  1B        DEC DE
116  40189  1B        DEC DE
117  40190  CDCOBB    CALL &&BBCO
118  40193  EB        EX HL,DE
119  40194  E3        EX HL,(SP)
120  40195  EB        EX HL,DE
121  40196  E5        PUSH HL
122  40197  D5        PUSH DE
123  40198  CDF6BB    CALL &BBF6
124  40201  D1        POP DE
125  40202  E1        POP HL
126  40203  C1        POP BC
127  40204  C9        RET

128  40205  E5        PUSH HL
129  40206  C5        PUSH BC
130  40207  ED42      SBC HL,BC
131  40209  C1        POP BC
132  40210  E1        POP HL
133  40211  C9        RET
```

Un des aspects les plus intéressants de cette routine est qu'elle détourne ce que l'on appelle un vecteur d'indirection. Ces vecteurs n'ont rien à voir avec les vecteurs d'appel déjà évoqués ; il s'agit simplement de sauts internes, donc utilisés par le système, et qui ne peuvent en aucun cas être appelés comme des sous-programmes. Néanmoins, et pour peu qu'on les connaisse, il est tout à fait possible de les détourner.

Celui auquel nous allons nous attaquer est situé à l'adresse &BDE8 et se présente en mémoire de la manière suivante (sur le 464 ; pour les deux autres machines, voir en fin de chapitre) :

PEEK(&BDE8) = &C3
PEEK(&BDE9) = &68
PEEK(&BDEA) = &0C

Une description précise du rôle de ce vecteur d'indirection ne présenterait guère d'intérêt. Disons simplement qu'il est utilisé pour l'écri-

ture d'un ou de plusieurs points sur l'écran ; lorsque le système "passe" par ce vecteur, HL contient l'adresse écran et B contient l'encre encodée.

Le but du détournement est tout simplement de modifier "au passage" la valeur contenue dans B en la remplaçant par le paramètre C. Ceci est effectué par les lignes 3 et 4 : l'adresse &0C68 contenue dans les emplacements mémoire &BDE9 et &BDEA est remplacée par l'adresse 40054, vers laquelle se fera donc le JUMP de l'adresse &BDE8 : c'est naturellement à cette adresse qu'est embusquée notre routine de remplacement (lignes 26 et 27). Cette routine est on ne peut plus simple : B est chargé avec une valeur donnée (vous vous doutez bien que l'emplacement mémoire 40055 ne reste pas à 0 : il est chargé avec le paramètre C par les lignes 5 et 6), après quoi le JUMP initial est effectué comme si de rien n'était. Il faut noter que le vecteur d'indirection est remis en état, lorsque la routine est terminée, par les lignes 23 et 24.

Incidemment, signalons que ce vecteur est utilisé entre autres par les instructions DRAW et DRAWR, et que le même détournement permet d'obtenir des tracés de lignes tout à fait particuliers. Cela devrait vous donner des idées...

L'organigramme de la partie du programme chargée de colorer le haut de la figure par rapport aux coordonnées transmises est le suivant :

1. A partir d'une position (X,Y) située dans la figure, chercher la position (XG,Y) la plus à gauche possible (c'est-à-dire jusqu'à la rencontre d'un point de délimitation de la figure).
2. Retenir cette position et chercher, sur la même ligne, la position (XD,Y) la plus à droite jusqu'à la rencontre d'un point de délimitation de la figure. Lorsque ce deuxième point est trouvé, tracer une ligne entre (XG,Y) et (XD,Y).
3. A partir de la position (XG,Y), essayer de monter d'une ligne. Si cela est possible, reprendre en 1. Si cela n'est pas possible (donc si un point de délimitation de la figure s'y trouve), se décaler d'un point vers la droite en vérifiant que XD n'est pas atteint (si c'est le cas, le haut de la figure est rempli) et essayer à nouveau.

Bien entendu, les vérifications adéquates de sortie d'écran doivent être faites lors de toutes ces opérations.

L'organigramme de la partie chargée de colorer le bas de la figure par rapport aux coordonnées transmises est exactement le même sauf que dans la troisième partie il faudra essayer de descendre au lieu d'essayer de monter.

Disons-le tout de suite, il est pratiquement impossible de fournir l'explicatif détaillé d'un programme de ce genre. Cela prendrait des pages et des pages, et il n'est même pas sûr que les choses en seraient plus claires pour autant. Nous nous contenterons donc d'en décrire les grandes lignes et de signaler les points particuliers. Pour le reste, une lecture attentive (et en surveillant tout particulièrement la pile qui est très largement utilisée) devrait suffire.

■ LE PROGRAMME PRINCIPAL (LIGNES 7 A 25)

Lignes 7 à 12

Chargement des registres HL et DE avec les paramètres X et Y, et préservation de ces paramètres sur la pile.

Lignes 13 à 15

Remplissage de la partie haute de la figure.

Lignes 16 à 22

Récupération des paramètres X et Y, et remplissage de la partie basse de la figure.

Lignes 24 et 25

Remise en état du vecteur &BDE9.

■ LES SOUS-PROGRAMMES

Sous-programme 1 (lignes 28 à 58)

Ce sous-programme est appelé après qu'une première ligne a été tracée d'une position (XG,Y) à une position (XD,Y) ; il boucle jusqu'à ce que toute la partie haute de la figure soit remplie. Si l'on exclut les appels à d'autres sous-programmes, sa tâche spécifique est celle qui est définie dans la partie 3 de l'organigramme général exposé plus haut. Il faut signaler que les lignes 38 et 39 vérifient que HL reste

inférieur ou égal à 399 et ne dépasse donc pas l'écran par le haut (le RET NC de la ligne 43 se rapporte bien sûr à cette vérification).

Sous-programme 2 (lignes 59 à 86)

Il est le pendant du précédent mais concerne le remplissage de la partie basse de la figure. Il s'agira donc cette fois d'essayer de descendre d'une ligne au lieu d'essayer de monter et la ligne 70 vérifie que HL reste supérieur ou égal à 0 (s'il devient négatif, son bit 7 est mis).

Sous-programme 3 (lignes 87 à 100)

A partir d'une position actuelle (X,Y), aller le plus possible à gauche. En entrée, HL est chargé avec Y et DE avec X ; en sortie, HL est inchangé et DE contient XG. La ligne 96 vérifie que DE reste > =0 et donc qu'il n'y a pas de sortie d'écran par la gauche.

Sous-programme 4 (lignes 101 à 127)

A partir d'une position (XG,Y) à l'extrême gauche, chercher le point (XD,Y), situé le plus possible à droite ; tracer ensuite une ligne entre (XD,Y) et (XG,Y). En entrée, HL contient Y et DE XG ; en sortie, HL et DE sont inchangés et BC contient XD. Les lignes 111 à 114 vérifient que DE reste inférieur à 640, et donc qu'il n'y a pas de sortie d'écran par la droite. (Si DE=639, le bit 7 de E=0 et le bit 1 de D=1 ; si DE=640, le bit 7 de E=1 et le bit 1 de D=1. Cela se vérifie facilement grâce à l'instruction BASIC BIN$.)

Sous-programme 5 (lignes 128 à 133)

Il teste si HL est plus petit que BC. Les registres sont inchangés en sortie et le sémaphore de report est vrai si HL < BC. Il est faux dans le cas contraire.

■ LES VECTEURS D'APPEL

&BBF0 Test d'un point de coordonnées absolues X et Y. En entrée, DE doit contenir X et HL doit contenir Y ; les registres BC, HL et DE sont modifiés en sortie et A contient le numéro de la couleur du point testé.

&BBC0 Positionnement du curseur graphique au point de coordonnées absolues X et Y. En entrée, DE doit contenir X et HL doit contenir Y ; tous les registres sont modifiés en sortie.

&BBF6 Traçage d'une ligne à partir de la position actuelle du curseur graphique jusqu'à un point de coordonnées absolues X et Y. En entrée, DE doit contenir X et HL doit contenir Y ; tous les registres sont modifiés en sortie.

Voici le programme de chargement BASIC (chaque fois qu'une touche est pressée, les figures de l'écran sont remplies avec des couleurs choisies aléatoirement entre 1 et 255. Pour arrêter, il faut donc utiliser la touche ESC).

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 16
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR EN
 LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE03C021769C22E9BD2373DD66,695
170 DATA 03DD6E02DD5605DD5E04E5D5CD,64E
180 DATA D29CCDE69CCD7B9CD1E12B2BCD,876
190 DATA D29CCDE69CCDA99C21680C22E9,76F
200 DATA BDC90600C3680CC5D52323E5CD,655
210 DATA F0BBFE002013E1E5019001ED42,663
220 DATA E1D1C1D0CDD29CCDE69C18E1E1,9A7
230 DATA 2B2BEBE1C12323CD0D9DEB38D3,696
240 DATA C9C5D52B2BE5CDF0BBFE00200E,742
250 DATA E1D1C1CB7CC0CDD29CCDE69C18,91C
260 DATA E6E12323EBE1C12323CD0D9DEB,742
270 DATA 38D8C9E51B1BD5CDF0BBD1E1FE,8F1
```

```
280 DATA 002004CB7A28EF1313C9D5E513,53C
290 DATA 13D5CDF0BBD1E1FE002008CB4A,74D
300 DATA 28EFCB7B28EB1B1BCDC0BBEBE3,7BC
310 DATA EBE5D5CDF6BBD1E1C1C9E5C5ED,AF6
320 DATA 42C1E1C9,2AD
330 '
340 '          DEMONSTRATION
350 '
360 MODE 1:REM (OU MODE 0, AU CHOIX)
370 C=0
380 FOR H=380 TO 120 STEP -130:C=C+1
390 PLOT 100,H,C:DRAW 150,H-50:DRAW 100,H-
100
400 DRAW 50,H-50:DRAW 100,H
410 PLOT 155,H:DRAW 245,H:DRAW 200,H-100:D
RAW 155,H
420 PLOT 305,H:DRAW 320,H-35:DRAW 360,H-50
:DRAW 320,H-65
430 DRAW 305,H-100:DRAW 290,H-65:DRAW 250,
H-50:DRAW 290,H-35:DRAW 305,H
440 PLOT 380,H:DRAW 450,H:DRAW 450,H-100:D
RAW 380,H-100:DRAW 380,H
450 PLOT 470,H:DRAW 580,H:DRAW 550,H-50:DR
AW 580,H-100:DRAW 470,H-100
460 DRAW 500,H-50:DRAW 470,H
470 NEXT H
480 '
490 '
500 FOR H=380 TO 120 STEP -130
510 FOR I=1 TO 5
520 COU=INT(RND*255)+1:READ X
530 CALL 40000,X,H-20,COU
540 DATA 100,170,305,420,510
550 NEXT I:RESTORE 540:NEXT H
560 REP$=INKEY$:IF REP$="" THEN 560
570 GOTO 360
```

■ ADAPTATION AU 664 ET AU 6128

Pour le 664

L'aspect en mémoire du vecteur d'indirection est le suivant :

PEEK(&BDE8) = &C3
PEEK(&BDE9) = &6D
PEEK(&BDEA) = &0C

Les lignes suivantes du programme de chargement BASIC devront par conséquent être modifiées :

```
190 DATA D29CCDE69CCDA99C216D0C22E9,774
200 DATA BDC90600C36D0CC5D52323E5CD,65A
```

Pour le 6128

L'aspect en mémoire du vecteur d'indirection est le suivant :

PEEK(&BDE8) = &C3
PEEK(&BDE9) = &71
PEEK(&BDEA) = &0C

Les lignes suivantes du programme de chargement BASIC devront être modifiées :

```
190 DATA D29CCDE69CCDA99C21710C22E9,778
200 DATA BDC90600C3710CC5D52323E5CD,65E
```

12 CARACTÈRES GÉANTS EN MODE 1 OU 2

Cette routine permet d'obtenir l'affichage des caractères du mode 0, alors même que l'on est en mode 1 ou 2. Qui plus est, ces caractères peuvent revêtir différents aspects (couleur et forme), selon les valeurs que l'on charge dans trois emplacements mémoire du programme.

Le format d'appel de cette routine est le suivant :

CALL 40000, NC, NL, @X$

- Le paramètre **@X$** représente l'adresse du descripteur de chaîne de la variable alphanumérique X$, qui doit donc avoir été définie avant l'appel (rappelons que la fonction du descripteur de variable @ a été vue lors de l'étude du Progamme 6).
- Les paramètres **NC** et **NL** indiquent les numéros de ligne et de colonne où l'on souhaite voir s'afficher la chaîne en caractères géants.

Il importe de préciser qu'une solide connaissance de l'organisation de la mémoire écran est indispensable à la bonne compréhension de ce programme (connaissance faisant d'ailleurs partie du minimum indispensable pour quiconque s'intéresse de près ou de loin à la programmation en langage machine ou en assembleur). Ce sujet a déjà été traité dans pas mal d'ouvrages consacrés au CPC et il y a toutes les chances pour que vous en possédiez un. Nous n'y reviendrons donc pas en détail ; rappelons simplement que :

1. Sauf cas bien particuliers (après un scrolling de ligne, par exemple), la mémoire écran débute à l'adresse 49152 (&C000) et finit à l'adresse 65535 (&FFFF).
2. Chacune des 25 lignes de l'écran est composée de 80 octets en largeur et de 8 octets en hauteur, cela indépendamment du mode.
3. Si l'on considère qu'une ligne est composée de 8 traits (8 séries de 80 octets les unes au-dessus des autres), la numérotation des

16 384 octets de la mémoire écran est organisée de la manière suivante :

- De 49152 à 51199 inclus : premier trait des 25 lignes (de 49152 à 49231 : 1er trait, 1re ligne ; de 49232 à 49311 : 1er trait, 2e ligne ; etc.).
- De 51200 à 53247 inclus : deuxième trait des 25 lignes.
- ... et ainsi de suite jusqu'à :
- de 63488 à 65535 inclus : huitième trait des 25 lignes.

Remarque

Si l'on vérifie tout cela par le calcul, on peut remarquer que, pour chacun des 8 traits, 48 octets sont "de trop" (pour le premier trait des 25 lignes, par exemple, il faut $80 \times 25 = 2\,000$ octets ; or de 40152 à 51199 inclus, il y a 2 048 octets). Ces 48 octets "superflus" n'apparaissent effectivement pas sur l'écran et sont utilisés par le système pour des choses particulières, par exemple les scrollings.

4. De cette organisation spécifique de la mémoire écran il ressort que :

- Pour passer, *sur une même ligne*, d'un trait à l'autre, il faut ajouter ou retrancher, par rapport à l'adresse où l'on se trouve, 2 048 ou l'un de ses multiples.
- Pour passer d'une situation dans une ligne *à une situation équivalente sur une autre ligne*, il faut ajouter ou retrancher 80 ou l'un de ses multiples.

Bien entendu, toute écriture directe en mémoire écran par l'intermédiaire du langage machine doit être soigneusement contrôlée par le programmeur. Une écriture intempestive qui sortirait des strictes limites de l'écran pourrait en effet avoir des conséquences fatales... pour le programme en cours, en tout cas.

Un éventuel dépassement de l'adresse 65535 est vérifié facilement : il s'agit en effet là de la valeur maximale que l'on peut charger dans un registre double. Si cette valeur est dépassée, cela engendre forcément un report, et il suffit donc de vérifier systématiquement le sémaphore de report.

La vérification dans l'autre sens peut se faire en testant l'octet fort du registre concerné : la valeur minimale légale est 49152, soit &C000

en hexadécimal ; dès que l'on passe à un chiffre inférieur, l'octet fort est modifié. Ainsi par exemple 49151 s'écrit &BFFF en hexadécimal. Il suffit donc de comparer le registre de poids fort avec &BF. Si la comparaison indique l'égalité, c'est que la limite légale est franchie et qu'il faut agir en conséquence.

La représentation des caractères dans ce contexte est également un sujet très intéressant, mais sur lequel on a déjà beaucoup écrit. Il nous suffira de rappeler qu'un caractère "mesure" 4 octets de large en mode 0, 2 octets en mode 1 et 1 octet en mode 2, leur hauteur étant toujours de 8 octets.

Les valeurs à écrire en mémoire écran pour représenter tel ou tel caractère dépendent bien sûr du caractère lui-même, mais également de sa couleur. Pour de plus amples informations, consultez votre bibliothèque informatique. Il y a fort à parier que vous y trouverez tous les renseignements nécessaires...

Ce programme va nous amener à explorer une zone de mémoire ROM bien particulière que l'on appelle la *table du générateur de caractères*. Dans cette table qui comporte 256×8 octets sont rangées les valeurs qu'il faut écrire en mémoire écran pour représenter, en mode 2, chacun des 256 caractères existants (les codes 0 à 31 sont des codes de contrôle, mais tous ont quand même une représentation graphique).

Toutes ces valeurs sont rangées par blocs de 8 octets (un caractère en mode 2 = 1 octet de large sur 8 de haut) et dans l'ordre ASCII. La table commence à l'adresse 14336 (&3800) par les 8 octets de la représentation graphique du code de contrôle 0, viennent ensuite les 8 octets de la représentation graphique du code de contrôle 1, etc. On peut donc en tirer cette conclusion que l'adresse dans la table du premier des 8 octets d'un caractère peut être calculée à partir de son code ASCII grâce à la formule suivante :

$$\text{adresse} = 14336 + (8 \times \text{code ASCII})$$

Un exemple concret permettra sans doute de mieux fixer les idées : le code ASCII de la lettre A est &41, ou 65 en décimal ; dans la table du générateur de caractères, l'adresse de l'image de ce caractère est $14336 + (8*65) = 14856$ (soit &3A08 en hexadécimal). Une lecture de cette adresse ROM et des 7 suivantes donne les résultats suivants (notez que l'instruction PEEK n'est pas utilisée. Elle ne permet pas, en effet, de lire en mémoire ROM) :

&3A08 → &18
&3A09 → &3C
&3A0A → &66
&3A0B → &66
&3A0C → &7E
&3A0D → &66
&3A0E → &66
&3A0F → &00

Essayez maintenant de lancer ce petit programme :

```
10 MODE 2
20 FOR i=49152 TO 63488 STEP 2048
25 READ a$:POKE i,VAL("&"+a$):NEXT
30 DATA 18,3C,66,66,7E,66,66,0
40 LOCATE 1,3
```

Les choses doivent être plus claires, non ?...

Nous savons donc maintenant comment trouver en mémoire ROM l'image d'un caractère donné. Il nous reste à résoudre le problème suivant : comment agrandir cette image avant écriture en mémoire écran pour obtenir des caractères identiques à ceux du mode 0 ?

Reprenons l'exemple du caractère A et, pour être encore plus précis, étudions le processus pour l'octet du haut (&18).

Les caractères du mode 2 ont un octet de large, alors que les caractères du mode 0 en ont 4 ; le rapport est donc de 4, ce qui revient à dire que l'image ROM devra être agrandie 4 fois.

Si l'on divise par deux chacun des 4 octets de l'image agrandie, il est possible d'obtenir une équivalence : le bit 7 de l'octet image correspondra à la moitié gauche du premier octet de l'image agrandie, le bit 6 de l'octet image à la moitié droite du premier octet de l'image agrandie, le bit 5 à la moitié gauche du deuxième octet, le bit 4 à la moitié droite du deuxième octet, etc. (rappelons que les bits d'un octet sont numérotés de 0 à 7 en partant de la droite).

Dans notre exemple, &18 s'écrit 00011000 en binaire ; pour une écriture agrandie en haut et à gauche de l'écran, cela donnera :

Bit 7 à 0 → moitié gauche de 49152 éteinte.
Bit 6 à 0 → moitié droite de 49152 éteinte.
Bit 5 à 0 → moitié gauche de 49153 éteinte.

Bit 4 à 1 → moitié droite de 49153 allumée.
Bit 3 à 1 → moitié gauche de 49154 allumée.
Bit 2 à 0 → moitié droite de 49154 éteinte.
Bit 1 à 0 → moitié gauche de 49155 éteinte.
Bit 0 à 0 → moitié droite de 49155 éteinte.

Attention

L'allumage d'un demi-octet dépend d'une part du mode dans lequel on se trouve, et d'autre part de la couleur que l'on souhaite voir s'afficher.

Exemples en mode 1 :

Poke 49152,192 met en jaune la moitié gauche de cette adresse écran.
Poke 49152,48 met en jaune la moitié droite.
Poke 49152,204 met en rouge la moitié gauche.

Comme nous l'avons dit, la compréhension de tout cela est subordonnée à une bonne connaissance de la mémoire écran. En tout état de cause, les valeurs adéquates peuvent à la limite être trouvées empiriquement.

Pour résumer ce paragraphe quelque peu rébarbatif, un caractère géant sera obtenu en testant les huit bits des huit octets de son image ROM, et en allumant au fur et à mesure, en mémoire écran, les demi-octets correspondants.

Voici maintenant le listing assembleur de cette routine.

```
1     40000  FE03      CP 3
2     40002  C0        RETNZ

3     40003  EB        EX HL,DE
4     40004  46        LD B,(HL)
5     40005  C5        PUSH BC
6     40006  23        INC HL
7     40007  5E        LD E,(HL)
8     40008  23        INC HL
9     40009  56        LD D,(HL)
10    40010  DD6604    LD H,(IX+4)
11    40013  DD6E02    LD L,(IX+2)
12    40016  2D        DEC H
```

```
13   40017  25          DEC L
14   40018  CD1ABC      CALL &BC1A
15   40021  C1          POP BC
16   40022  D5          PUSH DE
17   40023  CD669C      CALL 40038     *
18   40026  CD849C      CALL 40068     *
19   40029  110400      LD DE,4
20   40032  19          ADD HL,DE
21   40033  D1          POP DE
22   40034  13          INC DE
23   40035  10F1        DJNZ 40022
24   40037  C9          RET


25   40038  E5          PUSH HL
26   40039  1A          LD A,(DE)
27   40040  5F          LD E,A
28   40041  1600        LD D,0
29   40043  210000      LD HL,0
30   40046  3E08        LD A,8
31   40048  19          ADD HL,DE
32   40049  3D          DEC A
33   40050  20FC        JRNZ 40048
34   40052  7C          LD A,H
35   40053  C638        ADD A,56
36   40055  67          LD H,A
37   40056  EB          EX HL,DE
38   40057  E1          POP HL
39   40058  C9          RET


40   40059  DF          RST 3
41   40060  7F9C        40063          *
42   40062  C9          RET
43   40063  829C        40066          *
44   40065  FC          ROM select.
45   40066  1A          LD A,(DE)
46   40067  C9          RET


47   40068  C5          PUSH BC
48   40069  E5          PUSH HL
49   40070  CD7B9C      CALL 40059     *
```

```
50    40073   0604        LD B,4
51    40075   E5          PUSH HL
52    40076   CB7F        BIT 7,A
53    40078   2802        JRZ 40082
54    40080   36C0        LD (HL),192
55    40082   CB77        BIT 6,A
56    40084   2808        JRZ 40094
57    40086   3630        LD (HL),48
58    40088   CB7F        BIT 7,A
59    40090   2802        JRZ 40094
60    40092   36F0        LD (HL),240
61    40094   23          INC HL
62    40095   07          RLC A
63    40096   07          RLC A
64    40097   10E9        DJNZ 40076
65    40099   E1          POP HL
66    40100   7C          LD A,H
67    40101   C608        ADD A,8
68    40103   67          LD H,A
69    40104   3803        JRC 40109
70    40106   13          INC DE
71    40107   18D9        JR 40070
72    40109   E1          POP HL
73    40110   C1          POP BC
74    40111   C9          RET
```

Le programme est constitué d'une partie principale (lignes 1 à 24), et de trois sous-programmes. Voyons tout d'abord ces derniers :

■ SOUS-PROGRAMME 1 (LIGNES 25 A 39)

Il est chargé de calculer l'adresse, dans la table du générateur, d'un caractère donné (rappelons que la formule est 14336+(8×code ASCII). En entrée, DE doit pointer sur l'emplacement mémoire contenant le code du caractère ; en sortie, DE contient l'adresse en question et les autres registres sont inchangés.

Le code est d'abord chargé dans DE (lignes 26 à 28), puis la boucle 31-33 effectue le calcul 8×code ASCII (le registre A sert de compteur). Le résultat se retrouve dans HL et les lignes 34 à 36 additionnent 14336 à ce registre (en ajoutant 56 à son registre fort puisque

14336 = 256 × 56). Le résultat est ensuite chargé dans DE (ligne 37) et le retour effectué.

■ SOUS-PROGRAMME 2 (LIGNES 40 A 46)

Sous-programme minuscule, mais éminemment intéressant puisqu'il nous permet de voir comment il est possible de lire un emplacement mémoire de la ROM.

En entrée, DE contient l'adresse ROM ; en sortie, A est chargé avec le contenu de cet emplacement et les autres registres sont inchangés.

La lecture de la mémoire ROM doit se faire par l'intermédiaire d'un RESTART ; dans notre cas, il s'agit de &DF, ou RESTART 3, qui permet d'appeler une routine n'importe où en ROM ou en RAM. Ce RESTART doit être suivi de l'adresse d'un bloc composé de trois octets :

Octet 1 adresse de la routine qui doit être appelée (octet faible).

Octet 2 octet fort de cette même adresse.

Octet 3 octet de configuration, qui permet de sélectionner l'état RAM ou ROM. Dans le cas qui nous intéresse, l'octet &FC sélectionne la ROM BASIC, où se trouve notre fameuse table de caractères.

L'adresse qui suit le RESTART est 40063 ; comme il se doit, nous y trouvons d'abord l'adresse de notre routine suivie de l'octet de configuration. La routine elle-même est d'une simplicité presque comique : le registre A est simplement chargé avec la valeur de l'octet pointé par DE.

Répétons-le encore une fois, une lecture "ordinaire", sans passer par le RESTART ne permettrait de lire que la RAM, et en aucun cas la ROM.

Notons pour finir que deux RETURN sont nécessaires. Le premier (ligne 42) représente en quelque sorte le RETURN de notre sous-programme, et le second (ligne 46), le RETURN du RESTART.

■ SOUS-PROGRAMME 3 (LIGNES 47 A 74)

Ce sous-programme est véritablement le cœur de notre routine, puisqu'il réalise l'affichage sur écran d'un caractère géant. En entrée, HL doit pointer sur l'adresse écran à partir de laquelle devra être affi-

ché le caractère, et DE doit pointer sur l'adresse ROM de l'image du caractère dans la table du générateur ; en sortie, seuls HL et BC sont inchangés.

Le sous-programme est constitué de deux boucles imbriquées : la première (lignes 52 à 64) se charge de l'écriture d'un trait (n'oublions pas qu'un caractère a toujours 8 octets — ou traits — de haut, chacun d'entre eux ayant 4 octets de large). La seconde boucle (lignes 49 à 71) contrôle le nombre de traits qui ont été écrits et provoque le retour après le huitième. Voyons tout d'abord cette dernière boucle.

Le registre B, initialisé avec 4 en ligne 50, servira de compteur. La ligne 52 teste ensuite le bit 7 du registre A (qui contient un des 8 octets de l'image du caractère) ; deux cas sont alors possibles :

- *Si ce bit est à 0*, aucune opération particulière n'est à faire pour l'instant et le programme passe en ligne 55.
- *Si ce bit est à 1*, il faut alors allumer la moitié gauche de l'adresse écran pointée par HL ; cela est réalisé (ligne 54) par l'écriture de la valeur 192 dans cette adresse. (Cette valeur donnera un affichage en jaune. Il est bien évident qu'elle peut être modifiée pour donner d'autres couleurs, et même des mélanges de couleurs. Plusieurs exemples sont proposés dans le programme de chargement et démonstration situé en fin de chapitre.)

Dans les deux cas, le programme se poursuit en ligne 55, ou le bit 6 du registre A est testé à son tour. Cette fois encore, deux possibilités :

- *Si ce bit est à 0*, aucune opération particulière n'est à faire et le programme passe en ligne 40094.
- *Si ce bit est à 1*, les choses se compliquent un peu : la moitié droite de l'adresse écran pointée par HL est allumée en ligne 57, après quoi le bit 7 est de nouveau testé. La raison en est que le chargement de 48 dans l'adresse écran, tout en allumant la moitié droite de cette adresse, éteint en même temps la partie gauche (qui aurait éventuellement pu être allumée précédemment). Si ce nouveau test du bit 7 s'avère positif, il faudra donc allumer les deux moitiés de l'adresse écran concernée. Le cas échéant, cela est réalisé par la ligne 60.

Le registre HL est ensuite incrémenté (et pointe donc maintenant sur le deuxième des 4 octets, puis le registre A subit deux rotations

à gauche. Le résultat en est que le bit 5 de A se retrouve à la place du bit 7, et le bit 4 à la place du bit 6. Cela permet tout simplement de réutiliser la même partie du programme pour traiter successivement l'octet 1 par rapport aux bits 7 et 6, l'octet 2 par rapport aux bits 5 et 4, l'octet 3 par rapport aux bits 3 et 2 et l'octet 4 par rapport aux bits 1 et 0.

La ligne 64 contrôle la boucle, qui se termine lorsque les 4 octets ont été traités.

La boucle primaire commence en ligne 49 par l'appel du sous-programme 2, au retour duquel le registre A est chargé avec le contenu de l'un des 8 octets de l'image du caractère. Le compteur B de la boucle secondaire est ensuite initialisé, et HL préservé.

Le contrôle de sortie des 8 traits d'un caractère est effectué par les lignes 66 à 69, après que HL a été récupéré. Le processus de ce contrôle est extrêmement intéressant : au lieu d'un simple compteur initialisé à 8, c'est en effet la structure même de la mémoire écran qui est utilisée. Un exemple concret permettra une explication plus claire.

Imaginons que nous souhaitions afficher un caractère géant en haut et à gauche de l'écran. La zone de mémoire écran concernée se présente comme suit :

49152	49153	49154	49155
51200	51201	51202	51203
53248	53249	53250	53251
55296	55297	55298	55299
57344	57345	57346	57347
59392	59393	59394	59395
61440	61441	61442	61443
63488	63489	63490	63491

Pour passer d'un trait donné à celui qui est situé immédiatement en dessous, il faut ajouter 2048 à l'adresse actuelle (51200 = 49152 + 2048 ; 53248 = 51200 + 2048 ; etc.). Cette opération est réalisée à chaque tour de boucle par les lignes 66 à 68 (2048 = 256 × 8). Tout se passera bien jusqu'au moment où le programme va vouloir additionner 2048 à 63488. A ce moment-là, la valeur maximale que l'on peut charger dans un registre double (65535) sera dépassée, ce qui aura pour effet de positionner l'indicateur de report. Il suffira par conséquent de tester cet indicateur à chaque passe (ligne 69).

Si la boucle n'est pas terminée, DE est incrémenté de manière à pointer sur l'octet suivant de l'image du caractère, et le processus est repris à partir de la ligne 49.

Si la boucle est terminée, donc si les 8 traits du caractère sont affichés, les valeurs initiales de HL et BC sont récupérées sur la pile et le retour effectué.

■ LE PROGRAMME PRINCIPAL (LIGNES 1 A 24)

Le registre HL est pointé sur le descripteur de chaîne de la variable alphanumérique, dont la longueur est chargée dans B (ce registre servira de compteur à la boucle 16-23). Le registre DE est ensuite chargé avec l'adresse du premier caractère de la chaîne (lignes 6 à 9).

Le vecteur d'appel &BC1A permet d'obtenir l'adresse en mémoire écran d'un point dont on fournit l'abscisse et l'ordonnée (H doit contenir le numéro de colonne et L le numéro de ligne ; ces deux registres sont chargés en lignes 10 et 11). Notons que les coordonnées doivent être définies comme si les colonnes étaient numérotées de 0 à 39, et les lignes de 0 à 24 (ce qui explique les décrémentations des lignes 12 et 13).

La boucle 16-23 ne présente pas de particularité. Après qu'un caractère a été affiché par les sous-programmes 40038 et 40068, HL est pointé sur l'adresse écran où devra être affiché le suivant (lignes 19 et 20), et DE sur le prochain caractère de la chaîne (ligne 22).

Voici enfin le programme de chargement BASIC de cette routine qui fonctionne indifféremment sur les trois machines. (Notez les modifications des emplacements mémoire 40081, 40087 et 40093, et voyez le résultat chaque fois que vous pressez une touche. D'autres valeurs peuvent certainement être trouvées.) D'autre part, que se passerait-il selon vous si les JRZ des lignes 53, 56 et 59 du listing assembleur étaient transformés en JRNZ ?...

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 8
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
```

```
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR EN
 LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE03C0EB46C5235E2356DD6604,5F8
170 DATA DD6E022D25CD1ABCC1D5CD669C,6A7
180 DATA CD849C11040019D11310F1C9E5,5AE
190 DATA 1A5F16002100003E08193D20FC,268
200 DATA 7CC63867EBE1C9DF7F9CC9829C,857
210 DATA FC1AC9C5E5CD7B9C0604E5CB7F,7A6
220 DATA 280236C0CB7728083630CB7F28,46A
230 DATA 0236F023070710E9E17CC60867,4E4
240 DATA 38031318D9E1C1C9,3AA
250 '
260 '
270 '          DEMONSTRATION
280 '
290 A$="CARACTERES GEANTS"
300 B$="en mode 1"
310 FOR I=1 TO 13:READ X,Y,Z
320 POKE 40081,X:POKE 40087,Y:POKE 40093,Z
330 MODE 1:CALL 40000,3,11, @ A$
340 CALL 40000,9,14, @ B$
350 LOCATE 10,24:PRINT"PRESSEZ UNE TOUCHE"

360 REP$=INKEY$:IF REP$="" THEN 360
370 NEXT
380 DATA 192,48,240,12,3,15,204,51,255,192
,3,255,192,51,255,204,3,255,196,49
390 DATA 245,76,19,95,72,18,90,68,17,85,12
8,32,160,192,0,240,0,51,255
400 '
410 '
420 B$="en mode 2"
430 FOR I=1 TO 6:READ X,Y,Z
440 POKE 40081,X:POKE 40087,Y:POKE 40093,Z
450 MODE 2:CALL 40000,4,11, @ A$
460 CALL 40000,20,14, @ B$
470 LOCATE 30,24:PRINT"PRESSEZ UNE TOUCHE"
```

```
480 REP$=INKEY$:IF REP$="" THEN 480
490 NEXT
500 DATA 240,15,255,192,51,255,192,3,255,1
60,10,170,0,3,255,160,0,170
510 MODE 1
```

13 ADRESSE D'UNE LIGNE BASIC

Pour nombre de ses utilisateurs, le BASIC c'est un peu comme une voiture : on sait la conduire, on sait vaguement comment ça marche, mais on ne connaît finalement pas grand-chose de ce qu'il y a sous le capot.

Cela est certainement bien dommage : la découverte de l'organisation en RAM d'un programme BASIC est non seulement tout à fait passionnante, mais offre également d'intéressantes perspectives du point de vue de la programmation. C'est la raison pour laquelle nous vous proposons, dans ce chapitre et dans les deux qui suivent, une petite incursion dans ce que l'on pourrait presque appeler les ''coulisses du BASIC''.

La routine suivante a pour fonction d'affecter à une variable quelconque l'adresse RAM d'une ligne BASIC donnée. Alliée à une bonne connaissance des modalités de rangement d'un programme (qui peut être acquise grâce à de simples PEEK et une bonne dose de curiosité), elle peut considérablement faciliter l'écriture de programmes qui s'automodifient.

Son format d'appel est le suivant :

CALL 40000, NL, @ X

- **NL** indique le numéro de ligne dont on cherche l'adresse, et **X** est la variable à affecter (il est absolument nécessaire que cette variable soit créée avant l'appel pour éviter un *improper argument* ; il suffit pour cela d'un X=0...).
  Si une erreur est commise et que la ligne NL n'existe pas, un coup de cloche est envoyé et la main est rendue au BASIC sans complication d'aucune sorte.

Avant de voir le programme proprement dit, nous allons d'abord nous intéresser un peu au format de stockage d'une ligne BASIC, ou plus exactement aux cinq premiers des octets qui la composent :

Octet 1 il est toujours à 0 et sert de séparateur entre deux lignes.

Octet 2 octet faible de la longueur de la ligne.

Octet 3 octet fort de la longueur de la ligne.
Octet 4 octet faible du numéro de la ligne.
Octet 5 octet fort du numéro de la ligne.

Lorsque l'on parle de l'adresse d'une ligne BASIC, il s'agit en fait de l'adresse du séparateur.

Il faut savoir d'autre part qu'un programme BASIC est stocké à partir de l'adresse 367 (qui contiendra donc toujours le séparateur de la première ligne du programme, donc 0).

Tapez par exemple le petit programme suivant :

```
1 REM
2 PRINT
3 FOR I=367 TO 378 : PRINT PEEK(I):NEXT
```

Vous obtenez ceci :

```
  0 séparateur.
  6 octet faible de la longueur de la ligne 1.
  0 octet fort de la longueur.
  1 octet faible du numéro de la ligne 1.
  0 octet fort du numéro.
197 code de l'instruction REM (on dit aussi token de l'instruction).
  0 séparateur.
  6 octet faible de la longueur de la ligne 2.
  0 octet fort de la longueur.
  2 octet faible du numéro de la ligne 2.
  0 octet fort du numéro.
191 token de l'instruction PRINT.
```

L'adresse de la ligne 1 est 367, et l'adresse de la ligne 2 est 373.

Voici maintenant le listing assembleur de notre routine :

```
1     40000  D5          PUSH DE
2     40001  DD5603      LD D,(IX+3)
3     40004  DD5E02      LD E,(IX+2)
4     40007  CD589C      CALL 40024     *
5     40010  E1          POP HL
6     40011  2805        JRZ 40018
7     40013  EB          EX HL,DE
8     40014  CD40BD      CALL &BD40     #
9     40017  C9          RET
```

```
10    40018  3E07       LD A,7
11    40020  CD5ABB     CALL &BB5A
12    40023  C9         RET

13    40024  E5         PUSH HL
14    40025  216F01     LD HL,367
15    40028  CD769C     CALL 40054     *
16    40031  2002       JRNZ 40035
17    40033  E1         POP HL
18    40034  C9         RET
19    40035  E5         PUSH HL
20    40036  23         INC HL
21    40037  23         INC HL
22    40038  23         INC HL
23    40039  CD7F9C     CALL 40063     *
24    40042  E1         POP HL
25    40043  2803       JRZ 40048
26    40045  09         ADD HL,BC
27    40046  18EC       JR 40028
28    40048  3E02       LD A,2
29    40050  3D         DEC A
30    40051  EB         EX HL,DE
31    40052  E1         POP HL
32    40053  C9         RET

33    40054  23         INC HL
34    40055  4E         LD C,(HL)
35    40056  23         INC HL
36    40057  46         LD B,(HL)
37    40058  2B         DEC HL
38    40059  2B         DEC HL
39    40060  78         LD A,B
40    40061  B1         OR C
41    40062  C9         RET

42    40063  7E         LD A,(HL)
43    40064  BB         CP E
44    40065  C0         RET NZ
45    40066  23         INC HL
46    40067  7E         LD A,(HL)
47    40068  BA         CP D
48    40069  2B         DEC HL
49    40070  C9         RET
```

Voyons d'abord les trois sous-programmes.

■ SOUS-PROGRAMME 3 (LIGNES 42 A 49)

Sous-programme très simple et qui se passe presque de commentaire. Il est chargé d'effectuer la comparaison entre le nombre chargé dans DE et celui contenu dans les deux emplacements mémoire dont le premier est pointé par HL. En sortie, le registre A est modifié et l'indicateur de zéro (Z) n'est mis que si les deux nombres sont égaux.

■ SOUS-PROGRAMME 2 (LIGNES 33 A 41)

Il vérifie si la fin d'un programme BASIC est atteinte. En entrée, HL pointe sur ce qui serait le séparateur d'une ligne, en admettant que celle-ci existe (on pourrait le formuler autrement : HL pointe sur l'octet suivant immédiatement le dernier de la ligne précédente) ; en sortie, HL et DE sont inchangés. Si la ligne présupposée n'existe pas, donc si la fin du programme est atteinte, Z est mis et BC contient 0. Dans le cas contraire, Z n'est pas mis et BC contient la longueur de la ligne testée.

Ce sous-programme n'est pas non plus bien difficile à comprendre, lorsque l'on sait que la fin d'un programme BASIC est toujours signalée par cinq zéros successifs. Même si cela n'est pas très orthodoxe, on peut donc tout aussi bien admettre qu'il existe une dernière ligne dont la longueur et le numéro sont égaux à 0... une sorte de ligne fantôme. Il suffit donc de charger la longueur de ligne dans BC (lignes 33 à 36), et de tester ce registre (le OR C de la ligne 40 n'a pour résultat 0 que si B *et* C contiennent 0).

■ SOUS-PROGRAMME 3 (LIGNES 13 A 32)

Ce sous-programme, qui fait appel aux deux autres, est chargé de trouver l'adresse d'une ligne à partir de son numéro. En entrée, DE contient le numéro ; en sortie (et si la ligne existe), DE contient l'adresse, BC contient la longueur de la ligne, et Z n'est pas mis. Si la ligne n'existe pas, Z est mis. Dans tous les cas, HL est préservé. Il n'y a pas trente-six manières possibles d'effectuer cette recherche : le programme est parcouru ligne par ligne, jusqu'à ce que le numéro recherché soit trouvé.

Le registre HL pointant sur une adresse de ligne (au départ celle de la première ligne, donc 367), on vérifie d'abord que la fin du programme n'est pas atteinte (ligne 15). Si c'est le cas, donc si Z est vrai, le sous-programme s'arrête là ; dans le cas contraire, le saut de la ligne 16 est effectué vers la ligne 19. Là, HL est préservé, puis les trois décrémentations qui suivent le pointent sur l'octet faible du numéro de ligne. La comparaison entre ce numéro et celui contenu dans DE se fait par l'appel du sous-programme 3 (ligne 23). Deux cas sont alors possibles :

- *Si les deux numéros sont égaux*, l'adresse de la ligne en question est rechargée dans HL (ligne 24), et le programme passe en ligne 28. Les opérations des lignes 28 et 29 n'ont pour but que d'enlever l'indicateur Z (n'oublions pas qu'au retour de ce sous-programme cet indicateur ne doit être mis que si la ligne n'existe pas). L'adresse contenue dans HL est ensuite chargée dans DE (ligne 30), la valeur initiale de HL récupérée et le retour effectué.
- *Si les deux numéros sont différents*, il s'agit de pointer HL sur l'adresse de la ligne suivante avant de reprendre la boucle au début. Cela se fait très facilement en ajoutant la longueur de la ligne actuelle à son adresse, contenue dans HL (vous pouvez le vérifier dans l'exemple présenté plus haut). Le registre BC étant chargé avec la longueur de ligne (cela depuis l'appel, en ligne 15, du sous-programme 2), l'opération a lieu à la ligne 26. La boucle reprend ensuite en ligne 15.

■ LE PROGRAMME PRINCIPAL (LIGNE 1 A 9)

Le paramètre @ X est rangé sur la pile (ligne 1), puis DE est chargé avec le paramètre NL. Après la ligne 4, DE contient l'adresse de la ligne si elle existe (dans le cas contraire, le programme saute en ligne 10 : un coup de cloche est envoyé avant le retour au BASIC). Les contenus des registres HL et DE sont échangés en ligne 7 ; dès lors, HL contient l'adresse de la ligne et DE contient @ X.

Le vecteur &BD40 permet d'appeler une routine mathématique dont le rôle est de convertir un nombre en sa représentation en virgule flottante (donc stockée sur 5 octets). Avant l'appel, HL doit contenir le nombre, et DE doit pointer sur le premier des cinq emplacements mémoire de rangement (rappelons que les lignes d'un programme

BASIC peuvent être numérotées jusqu'à 65535, et que l'utilisation d'un format entier est par conséquent exclue).

Après la ligne 8, notre adresse est donc rangée à partir de l'adresse @ X, et notre routine est alors terminée.

Voici le programme de chargement BASIC (la démonstration a pour effet de remplacer les PRINT des lignes 310, 320 et 330 par des REM).

```
10 MODE 1:PRINT"CHARGEMENT EN COURS"
20 '
30 '
40 MEMORY 39999
50 ad=40000
60 FOR i=1 TO 6
70 READ a$,b$:v=0
80 FOR j=1 TO LEN(a$)-1 STEP 2
90 x=VAL("&"+MID$(a$,j,2)):v=v+x
100 POKE ad,x:ad=ad+1
110 NEXT j
120 IF v<>VAL("&"+b$) THEN PRINT"ERREUR
EN LIGNE ";160+(I-1)*10:END
130 NEXT i
140 '
150 '
160 DATA D5DD5603DD5E02CD589CE12805,617
170 DATA EBCD40BDC93E07CD5ABBC9E521,774
180 DATA 6F01CD769C2002E1C9E5232323,569
190 DATA CD7F9CE12803091 8EC3E023DEB,569
200 DATA E1C9234E23462B2B78B1C97EBB,605
210 DATA C0237EBA2BC9,30F
220 '
230 '
240 '           DEMONSTRATION
250 '
260 X=0:FOR I=310 TO 330 STEP 10
270 CALL 40000,I,àX
280 POKE X+5,197
290 NEXT I
300 '
310 PRINT
320 PRINT
330 PRINT
340 LIST 300-
```

■ ADAPTATION AU 664 ET AU 6128

Sur ces deux machines, l'adresse de la routine mathématique évoquée plus haut est &BD61. Une ligne du programme de chargement BASIC devra par conséquent être modifiée :

```
170 DATA EBCD61BDC93E07CD5ABBC9E521,795
```

14 INTERROGATION DU CLAVIER

La routine que nous vous proposons dans ce chapitre est en partie une application de ce qui a été vu au chapitre précédent. Elle permet l'interrogation du clavier et son format d'appel est le suivant :

CALL 40000, L1, L2, L3, L4

- Les quatre paramètres indiquent la ligne à laquelle doit sauter le programme BASIC en fonction de la touche curseur qui est pressée :

  Saut en ligne L1 si la touche pressée est ↑
  Saut en ligne L2 si la touche pressée est →
  Saut en ligne L3 si la touche pressée est ↓
  Saut en ligne L4 si la touche pressée est ←

Cette routine remplace donc les boucles d'interrogation classiques utilisant l'instruction INKEY. L'intérêt est double : d'abord un gain de place non négligeable, mais aussi et surtout une durée d'exécution pratiquement nulle.

Il est bien évident que n'importe quelle touche ou combinaison de touches peut être testée de la sorte ; comme en outre il est possible de transmettre jusqu'à 32 numéros de ligne...

Nous avons déjà évoqué, dans le Programme 5, le vecteur &BB1E permettant de tester si une touche est pressée. Il aurait bien sûr été possible de l'utiliser ; nous avons néanmoins choisi une autre solution, qui offre trois avantages ; elle est d'abord plus rapide, elle autorise ensuite des tests portant sur des combinaisons de touches (curseur vers le haut+COPY, par exemple), et elle va vous permettre de découvrir une zone RAM que l'on appelle la *table de scanning des touches*.

Il s'agit d'une zone de 10 octets (de l'adresse &B4EB à l'adresse &B4F4) dont les valeurs dépendent de la ou des touches qui sont pressées. (Attention: sur le 664 et le 6128, cette table commence en &B635 et finit en &B63E.)

Plutôt qu'un long discours, lancez le petit programme suivant (les possesseurs de 664 et 6128 devront modifier les adresses comme il a été dit plus haut). Pressez ensuite n'importe quelle touche ou com-

binaison de touches, et observez comment le contenu des emplacements mémoire se modifie.

```
10 CLS
20 FOR I=&B4EB TO &B4F4
30 PRINT PEEK(I);"";
40 NEXT:PRINT:PRINT
50 CALL &BB06:GOTO 20
```

Nous allons également nous intéresser à la variable système d'adresse &AE75 (&AE58 sur le 664 et le 6128), dans laquelle le contenu de HL est préservé avant qu'une instruction CALL soit exécutée (cette valeur est bien sûr remise dans HL au retour du programme machine).

Cette information n'aurait guère de quoi nous exciter s'il ne se trouvait que cette valeur est justement *celle de la ligne BASIC où devra reprendre le programme au retour de l'instruction CALL*. Vous avez certainement déjà compris : il suffira, au cours de notre routine, de modifier le contenu de ce placard en y mettant l'adresse de la ligne BASIC souhaitée, et le tour sera joué...

Voici le listing assembleur de cette routine :

```
1     40000  FE04       CP 4
2     40002  C0         RET NZ

3     40003  CDBA9C     CALL 40122     *
4     40006  283B       JRZ 40067
5     40008  ED53EF9C   LD (40175),DE *
6     40012  DD5603     LD D,(IX+3)
7     40015  DD5E02     LD E,(IX+2)
8     40018  CDBA9C     CALL 40122     *
9     40021  282C       JRZ 40067
10    40023  ED53ED9C   LD (40173),DE *
11    40027  DD5605     LD D,(IX+5)
12    40030  DD5E04     LD E,(IX+4)
13    40033  CDBA9C     CALL 40122     *
14    40036  281D       JRZ 40067
15    40038  ED53EB9C   LD (40171),DE *
16    40042  DD5607     LD D,(IX+7)
17    40045  DD5E06     LD E,(IX+6)
18    40048  CDBA9C     CALL 40122     *
```

```
19    40051   280E        JRZ 40067
20    40053   ED53E99C    LD (40169),DE *
21    40057   21409C      LD HL,40000   *
22    40060   3618        LD (HL),&18
23    40062   23          INC HL
24    40063   3647        LD (HL),&47
25    40065   1806        JR 40073

26    40067   3E07        LD A,7
27    40069   CD5ABB      CALL &BB5A
28    40072   C9          RET

29    40073   3AEBB4      LD A,(&B4EB)  #
30    40076   FE01        CP 1
31    40078   2815        JRZ 40101
32    40080   FE02        CP 2
33    40082   2818        JRZ 40108
34    40084   FE04        CP 4
35    40086   281B        JRZ 40115
36    40088   3AECB4      LD A,(&B4EC)  #
37    40091   FE01        CP 1
38    40093   C0          RET NZ

39    40094   2AEF9C      LD HL,(40175) *
40    40097   2275AE      LD (&AE75),HL #
41    40100   C9          RET

42    40101   2AE99C      LD HL,(40169) *
43    40104   2275AE      LD (&AE75),HL #
44    40107   C9          RET

45    40108   2AEB9C      LD HL,(40171) *
46    40111   2275AE      LD (&AE75),HL #
47    40114   C9          RET

48    40115   2AED9C      LD HL,(40173) *
49    40118   2275AE      LD (&AE75),HL #
50    40121   C9          RET


51    40122   E5          PUSH HL
52    40123   216F01      LD HL,367
53    40126   CDD89C      CALL 40152    *
```

```
54    40129  2002      JRNZ 40133
55    40131  E1        POP HL
56    40132  C9        RET
57    40133  E5        PUSH HL
58    40134  23        INC HL
59    40135  23        INC HL
60    40136  23        INC HL
61    40137  CDE19C    CALL 40161     *
62    40140  E1        POP HL
63    40141  2803      JRZ 40146
64    40143  09        ADD HL,BC
65    40144  18EC      JR 40126
66    40146  3E02      LD A,2
67    40148  3D        DEC A
68    40149  EB        EX HL,DE
69    40150  E1        POP HL
70    40151  C9        RET

71    40152  23        INC HL
72    40153  4E        LD C,(HL)
73    40154  23        INC HL
74    40155  46        LD B,(HL)
75    40156  2B        DEC HL
76    40157  2B        DEC HL
77    40158  78        LD A,B
78    40159  B1        OR C
79    40160  C9        RET

80    40161  7E        LD A,(HL)
81    40162  BB        CP E
82    40163  C0        RET NZ
83    40164  23        INC HL
84    40165  7E        LD A,(HL)
85    40166  BA        CP D
86    40167  2B        DEC HL
87    40168  C9        RET


88    40169  0000      Placard lig. 1
89    40171  0000      Placard lig. 2
90    40173  0000      Placard lig. 3
91    40175  0000      Placard lig. 4
```

Le programme est un peu particulier en ce sens qu'il se décompose en deux parties, dont l'une ne servira qu'une seule et unique fois, lors du premier appel. Il s'agit de la partie chargée de trouver les adresses des lignes dont les numéros ont été transmis comme paramètres. Dans un souci de rapidité, cette partie sera ensuite mise hors service. (Il serait absurde, en effet, que les adresses soient recherchées à chaque appel. Une fois trouvées, elles seront rangées dans les quatre placards situés en fin de programme.)

Cette mise hors service est réalisée de la manière suivante : lorsque les quatre adresses de ligne ont été trouvées et rangées dans leurs placards respectifs, le programme arrive à la ligne 21, où le registre HL est chargé avec l'adresse 40000. Les trois lignes suivantes ont ensuite pour effet d'installer un saut relatif de +71 en lieu et place du CP 4 du début de programme. Lors des appels suivants, le programme sautera donc directement à la ligne 29.

Remarque importante

Après le premier appel de la routine, aucune opération susceptible de modifier les adresses des lignes L1, L2, L3 et L4 (telle par exemple que l'insertion d'une nouvelle ligne) n'est en principe possible, puisque le programme de recherche des adresses en fonction des numéros est hors service et que les adresses erronées rangées dans les placards continueraient à être utilisées. Il existe en réalité deux parades simples :

- La première consiste à ne pas détruire le programme de chargement en mémoire de la routine, et à le laisser tel quel dans le programme BASIC. De cette manière, les emplacements mémoire 40000 et 40001 seront restaurés à chaque nouveau lancement du programme BASIC et toute modification éventuelle des adresses sera prise en compte.
- La seconde consiste à commencer le programme BASIC par la ligne suivante :

```
1 POKE 40000,&FE:POKE 40001,&04
```

Cela aura exactement le même effet.

Les trois sous-programmes utilisés pour la recherche des adresses de ligne (51-70 ; 71-79 ; 80-87) sont absolument identiques à ceux du chapitre précédent.

Au cas où une des lignes transmises n'existait pas, un saut est effectué en ligne 26 (coup de cloche et retour au BASIC).

L'interrogation du clavier (lignes 29 à 38) ne présente pas de difficulté particulière. Vous pouvez vérifier, en utilisant le programme BASIC proposé plus haut, que &B4EB contient 1 si l'on presse la touche ↑ , 2 si l'on presse la touche → , 4 si l'on presse la touche ↓ , et que &B4EC contient 1 si l'on presse la touche ← .

Selon le résultat du test, un saut est effectué vers les lignes 39, 42, 45 ou 48. L'adresse de la ligne correspondant à la touche pressée est chargée dans &AE75, après quoi l'on retourne au BASIC. Si aucune des quatre touches n'est pressée, le retour au BASIC est immédiat (ligne 38).

Voici le progarmme de chargement et de démonstration :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 13
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE04C0CDBA9C283BED53EF9CDD,7F0
170 DATA 5603DD5E02CDBA9C282CED53ED,63A
180 DATA 9CDD5605DD5E04CDBA9C281DED,668
190 DATA 53EB9CDD5607DD5E06CDBA9C28,6A0
200 DATA 0EED53E99C21409C3618233647,4BE
210 DATA 18063E07CD5ABBC93AEBB4FE01,5E6
220 DATA 2815FE022818FE04281B3AECB4,49C
230 DATA FE01C02AEF9C2275AEC92AE99C,731
240 DATA 2275AEC92AEB9C2275AEC92AED,6E4
250 DATA 9C2275AEC9E5216F01CDD89C20,681
260 DATA 02E1C9E5232323CDE19CE12803,650
270 DATA 0918EC3E023DEBE1C9234E2346,4F9
```

```
280 DATA 2B2B78B1C97EBBC0237EBA2BC9,690
290 DATA 0000000000000000,0
300 '
310 '
320 '           DEMONSTRATION
330 '
340 PRINT"PRESSEZ UNE TOUCHE CURSEUR"
350 CALL 40000,370,380,390,400
360 GOTO 350
370 PRINT CHR$(240);" ";:GOTO 350
380 PRINT CHR$(243);" ";:GOTO 350
390 PRINT CHR$(241);" ";:GOTO 350
400 PRINT CHR$(242);" ";:GOTO 350
```

■ ADAPTATION AU 664 ET AU 6128

Comme nous l'avons déjà dit, l'adresse &AE75 doit être remplacée par &AE58, et l'adresse &B4EB par &B635 (l'adresse &B4EC de la ligne 36 du programme assembleur devient naturellement &B636).

Les lignes suivantes devront être modifiées :

```
210 DATA 18063E07CD5ABBC93A35B6FE01,532
220 DATA 2815FE022818FE04281B3A36B6,3E8
230 DATA FE01C02AEF9C2258AEC92AE99C,714
240 DATA 2258AEC92AEB9C2258AEC92AED,6AA
250 DATA 9C2258AEC9E5216F01CDD89C20,664
```

15 RENUMÉROTATION ET DÉPLACEMENT DE BLOCS BASIC

La routine présentée dans ce chapitre est probablement la plus complexe de toutes, mais certainement aussi la plus intéressante. Ele constitue un complément fort utile de l'instruction RENUM, puisqu'elle permet le déplacement et la renumérotation d'un bloc de lignes précis. La pratique vous prouvera sans doute que cette pseudo-instruction est d'un intérêt incontestable, surtout si vous êtes amenés à mettre de l'ordre dans des programmes BASIC relativement longs.

Son format d'appel est le suivant :

CALL 40000, L1, L2, NL

- Les paramètres **L1** et **L2** représentent les bornes du bloc de lignes à déplacer, et **NL** le nouveau numéro de la ligne L1 (CALL 40000, 20, 45, 80 aurait par exemple pour effet de renuméroter le bloc de lignes 20 à 45 — la ligne 20 devenant la ligne 80 —, et de déplacer ce bloc dans le programme BASIC en fonction de sa nouvelle numérotation). Il convient de préciser que cette numérotation se fait de 1 en 1 (mais rien n'empêche ensuite d'utiliser un RENUM ordinaire...).

Bien entendu, l'ensemble du programme est examiné, et les numéros de ligne suivants des instructions telles que GOTO, GOSUB, RESTORE, etc. sont modifiés si nécessaire.

Pour un programme de ce genre, toutes les précautions doivent être prises, et un certain nombre d'autres vérifications sont effectuées pour éviter des désastres :

- Il faut que les lignes L1 et L2 existent. Sans commentaire !
- Il faut que L1 soit inférieur à L2.
- Il faut que le nombre total des octets constitutifs du bloc de lignes soit inférieur à 16384 (nous verrons pourquoi plus loin). Rassurez-vous, cela représente déjà un bon paquet de lignes.
- Il faut enfin que les nouveaux numéros de ligne n'interfèrent pas avec ceux de lignes déjà existantes.

Si l'une de ces erreurs se produit, un coup de cloche est envoyé et le retour se fait sans que le programme BASIC soit modifié en quoi que ce soit. Pas de risque donc, d'autant plus que nous avons testé cette routine de toutes les manières possibles.

Attention

Aucune vérification d'un éventuel dépassement du nombre 65535 par les nouveaux numéros de ligne n'a lieu. S'il est vrai qu'il s'agit d'une erreur susceptible d'être commise, avouons aussi qu'il faut vraiment le vouloir !...

Nous avons dit que des modifications de numéros de ligne internes devaient être envisagées (si par exemple le numéro de la ligne 100 est modifié par notre routine et qu'il y a un GOTO 100 quelque part dans le programme). Il est donc indispensable de bien connaître les modalités de rangement en RAM de ces numéros internes. Comme nous allons le voir, deux cas de figure sont possibles.

Lors de l'écriture du programme, les numéros internes sont stockés sous une forme ordinaire (deux octets), mais leur présence est marquée par le code &1E qui est placé juste devant. C'est ainsi par exemple qu'une ligne 5 GOTO 100 se présenterait comme ceci :

&00 séparateur.
&0A longueur, octet faible.
&00 longueur, octet fort.
&05 numéro, octet faible.
&00 numéro, octet fort.
&A0 code de l'instruction GOTO.
&20 code de l'espace.
&1E code indiquant qu'un numéro de ligne suit.
&64 octet faible de 100.
&00 octet fort de 100.

Mais cela n'est pas tout, car les concepteurs du CPC ont imaginé un système permettant un gain appréciable en temps d'exécution. Lorsque l'interpréteur BASIC rencontre un numéro de ligne, il est en effet obligé de rechercher cette ligne dans tout le programme ; cela prend évidemment du temps, surtout si l'on songe qu'une instruction comportant un numéro interne peut tout aussi bien être utilisée des centaines de fois (dans une boucle, par exemple).

En fait, cette recherche ne sera faite qu'une fois, parce qu'à la première rencontre de l'instruction en question le numéro de ligne sera remplacé par son adresse lorsque celle-ci aura été trouvée. Cette modification doit bien sûr être signalée, et le code &1E est remplacé par le code &1D qui indique qu'une adresse de ligne suit. Ainsi, si le programme repasse par là, il pourra directement sauter à l'adresse indiquée, sans plus avoir à effectuer la moindre recherche.

Notons que l'opération inverse (remplacement des adresses par les numéros) est parfois nécessaire, par exemple lorsque le programme doit être listé. Dans ce cas, en effet, ce sont les numéros et non pas les adresses qui doivent être affichés.

Tout cela va bien sûr nous être fort utile lorsque nous aurons à rechercher pour vérification les numéros internes du programme.

Avant d'en aborder l'étude détaillée, voyons d'abord le plan général de cette routine :

1. Remplacer toutes les adresses de ligne qui pourraient se trouver dans le programme par le *numéro de ligne* correspondant.
2. Calculer le nombre de lignes du bloc à renuméroter ainsi que le nouveau numéro de la dernière ligne ; vérifier qu'aucune interférence n'a lieu avec des lignes déjà existantes.
3. Calculer le nombre d'octets constitutifs du bloc et vérifier que ce nombre est inférieur ou égal à 16384.
4. Ranger provisoirement le bloc en mémoire écran (une zone mémoire tout à fait adéquate, toujours disponible, et qui offre un logement de 16 384 octets). Cela explique les étranges dessins que vous verrez s'afficher sur l'écran en cours d'exécution, et qui ne doivent pas vous effrayer... au contraire, c'est le signe que tout se passe bien.
5. Resserrer le reste du programme, de manière à combler la place ainsi laissée vacante.
6. Chercher à quel endroit doit être inséré le bloc en fonction des nouveaux numéros des lignes qui le constituent.
7. Ménager une place à cet endroit en "poussant" le programme et y insérer le bloc rangé en mémoire écran.
8. Modifier les numéros des lignes du bloc ainsi que tous les numéros internes du programme qui doivent l'être.

Voici le listing assembleur de cette routine :

```
1     40000  FE03       CP 3
2     40002  C0         RET NZ

3     40003  D5         PUSH DE
4     40004  216F01     LD HL,367
5     40007  23         INC HL
6     40008  CDAE9D     CALL 40366     *
7     40011  2821       JRZ 40046
8     40013  3E1D       LD A,&1D
9     40015  BE         CP (HL)
10    40016  20F5       JRNZ 40007
11    40018  2B         DEC HL
12    40019  7E         LD A,(HL)
13    40020  23         INC HL
14    40021  FE20       CP &20
15    40023  2804       JRZ 40029
16    40025  FE2C       CP &2C
17    40027  20EA       JRNZ 40007
18    40029  361E       LD (HL),&1E
19    40031  23         INC HL
20    40032  5E         LD E,(HL)
21    40033  23         INC HL
22    40034  56         LD D,(HL)
23    40035  EB         EX HL,DE
24    40036  D5         PUSH DE
25    40037  CD4A9D     CALL 40266     *
26    40040  E1         POP HL
27    40041  72         LD (HL),D
28    40042  2B         DEC HL
29    40043  73         LD (HL),E
30    40044  18D9       JR 40007

31    40046  DD6605     LD H,(IX+5)
32    40049  DD6E04     LD L,(IX+4)
33    40052  DD5603     LD D,(IX+3)
34    40055  DD5E02     ID E,(IX+2)
35    40058  CD0D9D     CALL 40205     *
36    40061  2826       JRZ 40101
37    40063  EB         EX HL,DE
38    40064  F1         POP AF
39    40065  E5         PUSH HL
```

```
40   40066   C5          PUSH BC
41   40067   F5          PUSH AF
42   40068   010000      LD BC,0
43   40071   03          INC BC
44   40072   C5          PUSH BC
45   40073   CDA59D      CALL 40357     *
46   40076   09          ADD HL,BC
47   40077   C1          POP BC
48   40078   E5          PUSH HL
49   40079   ED52        SBC HL,DE
50   40081   E1          POP HL
51   40082   20F3        JRNZ 40071
52   40084   E1          POP HL
53   40085   E5          PUSH HL
54   40086   09          ADD HL,BC
55   40087   C1          POP BC
56   40088   CDB89D      CALL 40376     *
57   40091   C1          POP BC
58   40092   E1          POP HL
59   40093   3007        JRNC 40102
60   40095   AF          XOR A
61   40096   3E3F        LD A,&3F
62   40098   90          SUB B
63   40099   3007        JRNC 40108
64   40101   C1          POP BC
65   40102   3E07        LD A,7
66   40104   CD5ABB      CALL &BB5A
67   40107   C9          RET

68   40108   C5          PUSH BC
69   40109   E5          PUSH HL
70   40110   1100C0      LD DE,49152
71   40113   EDB0        LDIR
72   40115   EB          EX HL,DE
73   40116   2A83AE      LD HL,(&AE83)  #
74   40119   ED52        SBC HL,DE
75   40121   23          INC HL
76   40122   E5          PUSH HL
77   40123   C1          POP BC
78   40124   E1          POP HL
79   40125   EB          EX HL,DE
80   40126   EDB0        LDIR
81   40128   1B          DEC DE
```

```
82   40129  DD4601     LD B,(IX+1)
83   40132  DD4E00     LD C,(IX+0)
84   40135  CD539D     CALL 40275     *
85   40138  C5         PUSH BC
86   40139  E1         POP HL
87   40140  C1         POP BC

88   40141  C5         PUSH BC
89   40142  D5         PUSH DE
90   40143  EB         EX HL,DE
91   40144  ED52       SBC HL,DE
92   40146  23         INC HL
93   40147  E5         PUSH HL
94   40148  C1         POP BC
95   40149  2A83AE     LD HL,(&AE83)  #
96   40152  D1         POP DE
97   40153  EB         EX HL,DE
98   40154  EDB8       LDDR
99   40156  23         INC HL
100  40157  C1         POP BC
101  40158  E5         PUSH HL
102  40159  1100C0     LD DE,49152
103  40162  EB         EX HL,DE
104  40163  EDB0       LDIR
105  40165  E1         POP HL

106  40166  DD4601     LD B,(IX+1)
107  40169  DD4E00     LD C,(IX+0)
108  40172  CD4A9D     CALL 40266     *
109  40175  E5         PUSH HL
110  40176  23         INC HL
111  40177  23         INC HL
112  40178  23         INC HL
113  40179  CDA09D     CALL 40352     *
114  40182  CD739D     CALL 40307     *
115  40185  DD6603     LD H,(IX+3)
116  40188  DD6E02     LD L,(IX+2)
117  40191  ED52       SBC HL,DE
118  40193  E1         POP HL
119  40194  C8         RET Z
120  40195  C5         PUSH BC
121  40196  CDA59D     CALL 40357     *
122  40199  09         ADD HL,BC
```

```
123   40200   C1          POP BC
124   40201   C8          RET Z
125   40202   03          INC BC
126   40203   18DF        JR 40172
```

Sous-programme 1

```
127   40205   CD2C9D      CALL 40236     *
128   40208   C8          RET Z
129   40209   EB          EX HL,DE
130   40210   CD2C9D      CALL 40236     *
131   40213   C8          RET Z
132   40214   23          INC HL
133   40215   4E          LD C,(HL)
134   40216   23          INC HL
135   40217   46          LD B,(HL)
136   40218   2B          DEC HL
137   40219   2B          DEC HL
138   40220   E5          PUSH HL
139   40221   ED52        SBC HL,DE
140   40223   3808        JRC 40233
141   40225   09          ADD HL,BC
142   40226   44          LD B,H
143   40227   4D          LD C,L
144   40228   E1          POP HL
145   40229   3E02        LD A,2
146   40231   3D          DEC A
147   40232   C9          RET
148   40233   E1          POP HL
149   40234   AF          XOR A
150   40235   C9          RET
```

Sous-programme 2

```
151   40236   E5          PUSH HL
152   40237   216F01      LD HL,367
153   40240   CDA59D      CALL 40357     *
154   40243   2002        JRNZ 40247
155   40245   E1          POP HL
156   40246   C9          RET
157   40247   E5          PUSH HL
```

```
158   40248   23          INC HL
159   40249   23          INC HL
160   40250   23          INC HL
161   40251   CD989D      CALL 40344     *
162   40254   E1          POP HL
163   40255   2803        JRZ 40260
164   40257   09          ADD HL,BC
165   40258   18EC        JR 40240
166   40260   3E02        LD A,2
167   40262   3D          DEC A
168   40263   EB          EX HL,DE
169   40264   E1          POP HL
170   40265   C9          RET
```

Sous-programme 3

```
171   40266   E5          PUSH HL
172   40267   23          INC HL
173   40268   23          INC HL
174   40269   23          INC HL
175   40270   5E          LD E,(HL)
176   40271   23          INC HL
177   40272   56          LD D,(HL)
178   40273   E1          POP HL
179   40274   C9          RET
```

Sous-programme 4

```
180   40275   D5          PUSH DE
181   40276   216F01      LD HL,367
182   40279   CD4A9D      CALL 40266     *
183   40282   E5          PUSH HL
184   40283   C5          PUSH BC
185   40284   E1          POP HL
186   40285   ED52        SBC HL,DE
187   40287   E1          POP HL
188   40288   380C        JRC 40302
189   40290   C5          PUSH BC
190   40291   CDA59D      CALL 40357     *
191   40294   C5          PUSH BC
192   40295   D1          POP DE
```

```
193   40296   C1          POP BC
194   40297   2803        JRZ 40302
195   40299   19          ADD HL,DE
196   40300   18E9        JR 40279
197   40302   AF          XOR A
198   40303   E5          PUSH HL
199   40304   C1          POP BC
200   40305   D1          POP DE
201   40306   C9          RET
```

Sous-programme 5

```
202   40307   E5          PUSH HL
203   40308   216F01      LD HL,367
204   40311   23          INC HL
205   40312   CDAE9D      CALL 40366      *
206   40315   2002        JRNZ 40319
207   40317   E1          POP HL
208   40318   C9          RET
209   40319   3E1E        LD A,&1E
210   40321   BE          CP (HL)
211   40322   20F3        JRNZ 40311
212   40324   2B          DEC HL
213   40325   7E          LD A,(HL)
214   40326   23          INC HL
215   40327   FE20        CP &20
216   40329   2804        JRZ 40335
217   40331   FE2C        CP &2C
218   40333   20E8        JRNZ 40311
219   40335   23          INC HL
220   40336   CD989D      CALL 40344      *
221   40339   CCA09D      CALLZ40352      *
222   40342   18DF        JR 40311
```

Sous-programme 6

```
223   40344   7E          LD A,(HL)
224   40345   BB          CP E
225   40346   C0          RET NZ
226   40347   23          INC HL
227   40348   7E          LD A,(HL)
```

```
228   40349   BA          CP D
229   40350   2B          DEC HL
230   40351   C9          RET
```

Sous-programme 7

```
231   40352   71          LD (HL),C
232   40353   23          INC HL
233   40354   70          LD (HL),B
234   40355   2B          DEC HL
235   40356   C9          RET
```

Sous-programme 8

```
236   40357   23          INC HL
237   40358   4E          LD C,(HL)
238   40359   23          INC HL
239   40360   46          LD B,(HL)
240   40361   2B          DEC HL
241   40362   2B          DEC HL
242   40363   78          LD A,B
243   40364   B1          OR C
244   40365   C9          RET
```

Sous-programme 9

```
245   40366   EB          EX HL,DE
246   40367   E5          PUSH HL
247   40368   2A83AE      LD HL,(&AE83) #
248   40371   ED52        SBC HL,DE
249   40373   E1          POP HL
250   40374   EB          EX HL,DE
251   40375   C9          RET
```

Sous-programme 10

```
252   40376   E5          PUSH HL
253   40377   C5          PUSH BC
254   40378   C5          PUSH BC
```

```
255  40379  D1         POP DE
256  40380  CD2C9D     CALL 40236     *
257  40383  2015       JRNZ 40406
258  40385  C1         POP BC
259  40386  CD539D     CALL 40275     *
260  40389  C5         PUSH BC
261  40390  E1         POP HL
262  40391  CDA59D     CALL 40357     *
263  40394  2003       JRNZ 40399
264  40396  37         SCF
265  40397  E1         POP HL
266  40398  C9         RET
267  40399  CD4A9D     CALL 40266     *
268  40402  E1         POP HL
269  40403  ED52       SBC HL,DE
270  40405  C9         RET
271  40406  C1         POP BC
272  40407  E1         POP HL
273  40408  AF         XOR A
274  40409  C9         RET
```

Dans un souci de clarté, un maximum de sous-programmes ont été mis en place. Cela permet d'avoir une vision beaucoup plus nette de la démarche générale. Nous allons les étudier dans le désordre, en commençant par ceux qui n'en appellent pas d'autres.

■ SOUS-PROGRAMME 3 (LIGNES 171 A 179)

Il est chargé de trouver le numéro d'une ligne à partir de son adresse. En entrée, HL contient l'adresse ; en sortie, DE contient le numéro et tous les autres registres sont préservés. Son fonctionnement s'explique de lui-même.

■ SOUS-PROGRAMME 6 (LIGNES 223 A 230)

Il est le même que le sous-programme 3 du Programme 13 (s'y reporter pour plus de précisions). Rappelons simplement qu'il vérifie si le nombre pointé par HL est égal au nombre contenu dans DE. Le registre AF est modifié en sortie et l'indicateur Z n'est vrai que si les nombres sont égaux.

■ SOUS-PROGRAMME 7 (LIGNES 231 A 235)

Il remplace le nombre pointé par HL par celui contenu dans BC. Tous les registres sont préservés.

■ SOUS-PROGRAMME 8 (LIGNES 236 A 244)

Il est identique au sous-programme 2 du Programme 13 (s'y reporter) et vérifie si la fin du programme est atteinte. Si c'est le cas, l'indicateur Z est vrai en sortie.

■ SOUS-PROGRAMME 9 (LIGNES 245 A 251)

Ce sous-programme compare l'adresse contenue dans HL et celle contenue dans l'emplacement mémoire &AE83. Cet emplacement contient une variable système indiquant l'adresse de la fin du programme BASIC. Il s'agit donc là d'une autre manière de tester si la fin du programme est atteinte.

Sur le 664 et le 6128, cette variable est rangée en &AE66.

■ SOUS-PROGRAMME 2 (LIGNES 151 A 170)

Identique au sous-programme 1 du Programme 13 (s'y reporter), il est chargé de trouver l'adresse d'une ligne à partir de son numéro. En entrée, DE contient le numéro ; en sortie, et si la ligne existe, DE contient son numéro, BC sa longueur, et l'indicateur Z n'est pas mis. Si la ligne n'existe pas, Z est mis. Le registre HL est préservé dans les deux cas.

■ SOUS-PROGRAMME 1 (LIGNES 127 A 150)

Il calcule le nombre d'octets compris dans le bloc de lignes L1-L2. En entrée, HL contient L1 et DE contient L2 ; en sortie, HL contient l'adresse de la ligne L2, DE celle de la ligne L1, et BC le nombre d'octets. D'autre part, l'indicateur Z est mis si une des deux lignes n'existe pas ou si L1 est plus grand que L2.

Après les lignes 127 à 131, DE contient AD1 (adresse de la ligne L1) et HL contient AD2 (adresse de la ligne L2). Si l'une des deux lignes n'existe pas, le retour est immédiat (ligne 128 ou 131).

Le nombre d'octets du bloc est égal à AD2 – AD1 + longueur de L2. Le registre BC est d'abord chargé avec la longueur de L2 (lignes 132 à 135) ; la ligne 139 effectue ensuite la soustraction AD2 – AD1 dont le résultat se retrouve dans HL (si AD2 est plus grand que AD1, le saut de la ligne 140 vers l'adresse 40233 a lieu. Le XOR A de la ligne 149 n'est là que pour rendre vrai l'indicateur Z et signaler qu'il y a un problème). Le registre BC est finalement additionné au registre HL (ligne 141), et le résultat passe dans BC (lignes 142 et 143).

L'adresse AD2 est récupérée sur la pile (ligne 144) puis l'indicateur Z est enlevé (lignes 145 et 146).

■ SOUS-PROGRAMME 4 (LIGNES 180 A 201)

Ce sous-programme est appelé alors que le bloc à renuméroter est rangé en mémoire écran et que le reste du programme a été resserré (voir plan général). Il est chargé de rechercher l'adresse de la première ligne dont le numéro est plus grand que NL. Cette adresse sera bien sûr celle où devra être réinséré le bloc renuméroté.

En entrée, BC contient NL ; en sortie, BC contient l'adresse recherchée et HL est modifié. Si le bloc doit être installé à la fin du programme, BC contient l'adresse suivant immédiatement celle du dernier octet du programme.

La boucle de recherche commence en ligne 182 : HL pointant sur une adresse de ligne (367 au départ), l'appel du sous-programme 3 a pour effet de charger DE avec le numéro de cette ligne. La comparaison entre BC et DE est faite par les lignes 184 à 186 (le numéro contenu dans BC passe dans HL, et la soustraction de la ligne 186 engendrera un report si DE est plus grand que HL). Deux cas possibles, donc :

- Si DE > HL, la ligne recherchée est trouvée et le programme saute à l'adresse 40302 (le XOR A, que nous avons déjà utilisé pour mettre l'indicateur Z, a également pour effet d'enlever l'indicateur de report et c'est dans ce but qu'il est utilisé ici. Si cela n'était pas fait, cet indicateur jouerait un rôle tout à fait imprévu dans la suite du programme principal ; cas typique des pièges vicieux que tend parfois le langage machine et sur lesquels on peut se casser les dents un bon bout de temps).
  L'adresse, qui est contenue dans HL, passe dans BC (lignes 198 et 199), la valeur initiale de DE est récupérée et le retour effectué.

- Si DE n'est pas plus grand que HL, donc si aucun report n'est engendré, le programme passe en ligne 189 : BC est d'abord préservé, puis l'appel du sous-programme 8 vérifie si la fin du programme est atteinte (rappelons qu'au retour BC contient la longueur de la ligne testée ; revoir éventuellement l'explicatif de ce sous-programme). Dans tous les cas, la longueur de ligne passe dans DE (lignes 191 et 192), et la valeur initiale de BC est récupérée (ligne 193). Le test relatif à l'appel du sous-programme 8 a lieu à la ligne 194 : si la fin du programme est atteinte, donc si Z est vrai, l'adresse contenue dans HL (= adresse suivant celle du dernier octet du programme) est celle à partir de laquelle devra être installé le bloc renuméroté et le programme passe en ligne 197. Dans le cas contraire, HL est pointé sur l'adresse de la ligne suivante (adresse ligne suivante = adresse ligne actuelle + longueur ligne actuelle) et la boucle reprend en ligne 182.

■ SOUS-PROGRAMME 5 (LIGNES 202 A 222)

Ce sous-programme examine l'ensemble du programme BASIC et remplace tous les numéros internes X par le numéro Y. En entrée, DE contient X et BC contient Y ; en sortie, seul AF est modifié.

La recherche s'effectue de la manière suivante : HL pointe successivement sur tous les octets, qui sont comparés avec le code &1E annonciateur, en principe, d'un numéro de ligne. Deux précautions valant mieux qu'une et pour être sûr qu'il ne s'agit pas là d'une simple variable numérique, deux tests ont alors lieu, qui vérifient que l'octet précédant &1E est soit le code de l'espace, soit le code de la virgule. En effet, s'il s'agit véritablement d'un numéro de ligne, il est obligatoire que l'on trouve l'une ou l'autre de ces valeurs :

- Dans le cas, par exemple d'un simple GOTO X, ce sera le code de l'espace (si cet espace n'est pas mis lors de la saisie, un "Syntax error" est envoyé).
- Dans le cas d'un ON variable GOTO X, X1, X2, etc., ce sera bien sûr le code de la virgule.

La boucle de recherche commence en ligne 204, après que HL a été chargé avec l'adresse de début du programme. L'appel (en ligne 205) du sous-programme 9 vérifie si la fin du programme est atteinte.

Si c'est le cas, le retour se fait après que la valeur initiale de HL a été récupérée (lignes 207 et 208). Dans le cas contraire, la comparaison est faite avec &1E (ligne 210).

Si l'octet testé est différent de &1E, la boucle de recherche reprend en ligne 204 (HL pointe sur l'octet suivant et le processus est repris).

Si l'octet testé est égal à &1E, le contenu de l'octet précédent est chargé dans A (lignes 212 et 213) et l'on vérifie que A est égal soit au code de l'espace (ligne 215), soit au code de la virgule (ligne 217).

Si l'une des deux égalités est vérifiée, alors on peut être sûr que le nombre suivant le code &1E est bien un numéro de ligne. Dans ce cas, HL est pointé dessus (ligne 219) et l'appel du sous-programme 6 compare le numéro trouvé avec celui contenu dans DE. Le cas échéant (si Z est vrai au retour), le numéro est remplacé par celui contenu dans BC (appel conditionnel du sous-programme 7, en ligne 221). La boucle reprend ensuite en ligne 204.

■ SOUS-PROGRAMME 10 (LIGNES 252 A 274)

Ce dernier sous-programme est chargé d'examiner tout le programme BASIC pour vérifier qu'aucune interférence n'a lieu entre les nouveaux numéros du bloc à transférer et les numéros des lignes déjà existantes. En entrée, HL contient NL1 (nouveau numéro de la première ligne du bloc et qui se confond, en fait, avec le paramètre NL ; l'appellation NL1 ne sert qu'à bien le différencier de NL2), et BC contient NL2 (nouveau numéro de la dernière ligne du bloc). Tous les registres sont modifiés en sortie et l'indicateur de report n'est vrai que si aucune interférence n'est possible.

La démarche est la suivante :

1. Vérifier que NL1 n'existe pas déjà dans le programme. Si elle existe, enlever l'indicateur de report et revenir immédiatement. Si elle n'existe pas, passer à la deuxième phase.
2. Chercher l'adresse de la ligne qui suivra immédiatement le bloc renuméroté lorsqu'il aura été réinséré dans le programme. Si cette adresse est en fait située à la fin du programme, pas de problème ; on peut donc mettre l'indicateur de report et effectuer le retour. Si cette adresse est à l'intérieur du programme, vérifier que le numéro de la ligne correspondante est supérieur à NL2. Si tel est le cas, aucune interférence n'aura lieu.

Le registre DE est chargé avec NL1 (lignes 254 et 255) avant l'appel du sous-programme 2 (rappelons qu'au retour de ce sous-programme, Z n'est vrai que si la ligne dont le numéro est dans DE n'existe pas).

Si la ligne a été trouvée, donc si Z est faux, le saut de la ligne 257 est effectué vers la ligne 271 (nettoyage de la pile, suppression de l'indicateur de report pour signaler l'interférence au programme principal, et retour).

Si la ligne n'existe pas, le registre BC est chargé avec NL1 (ligne 258), puis avec *l'adresse de la ligne qui suivra immédiatement le bloc renuméroté lorsque celui-ci aura été réinséré* (cela grâce à l'appel du sous-programme 4). Cette adresse passe ensuite dans HL (lignes 260 et 261), et l'appel du sous-programme 8 permet de vérifier si cette adresse est celle de la fin du programme.

Si c'est le cas, aucune interférence n'est à craindre et le programme passe en ligne 264 (positionnement de l'indicateur de report, nettoyage de la pile et retour).

Dans le cas contraire, le numéro de cette ligne (dont l'adresse est toujours dans HL) est chargé dans DE (ligne 267), HL est chargé avec NL2 (ligne 268) et la comparaison est effectuée entre ces deux numéros (ligne 269). L'indicateur de report sera de toute façon positionné en fonction du résultat et le retour peut donc être effectué.

Passons maintenant au programme principal, qui ne devrait guère présenter de difficultés si les sous-programmes ont été bien compris :

Lignes 3 à 29

Examen du programme et remplacement de toutes les adresses de ligne rencontrées par le numéro correspondant (notons que les codes &1D sont remplacés par &1E).

Lignes 31 à 35

Chargement des paramètres L1 et L2 dans les registres HL et DE, puis chargement de DE avec AD1 et de HL avec AD2 (appel du sous-programme 1). Par la même occasion, l'existence de ces deux lignes est vérifiée ; le cas échéant, un saut est effectué vers le programme d'erreur (sortie d'un coup de cloche et abandon de la routine) des lignes 65 à 67.

Lignes 37 à 51

Calcul du nombre de lignes du bloc à renuméroter moins 1, cette valeur se retrouvant finalement dans BC. Le registre HL pointe au

départ sur AD1 et passe de ligne en ligne jusqu'à la rencontre de l'adresse AD2, contenue dans DE (la comparaison a lieu en ligne 49) ; à chaque passe, BC est incrémenté.

Lignes 52 à 55

Calcul de NL2 (nouveau numéro de la dernière ligne du bloc). Après ces lignes, HL contient NL2 et BC contient NL1.

Lignes 56 à 63

Calcul du nombre d'octets du bloc et vérification du non-dépassement de la valeur 16384 (en cas de dépassement, le saut de la ligne 63 n'a pas lieu, et le programme d'erreur est exécuté). A la fin de ces lignes, BC contient le nombre d'octets et HL contient l'adresse de la première ligne du bloc.

Lignes 70 et 71

Transfert du bloc en mémoire écran. Après ces lignes, HL contient l'adresse suivant immédiatement celle du dernier octet du bloc qui vient d'être transféré (ou, si l'on préfère, l'adresse de la ligne suivant immédiatement la dernière du bloc).

Lignes 72 à 77

Calcul du nombre d'octets de la partie du programme située *après* le bloc transféré. Cette partie devra être "recollée" à la partie du programme *précédant* le bloc transféré.

Lignes 78 à 80

Transfert de la fin du programme, de manière que la place laissée vacante par le bloc soit comblée. La ligne 81 pointe ensuite DE sur ce qui peut être considéré comme la fin du programme ainsi resserré.

Lignes 82 à 87

Chargement des registres, en prévision de la suite du programme (après ces lignes, HL contient l'adresse à partir de laquelle le bloc renuméroté devra être réinséré, DE contient l'adresse de fin du programme resserré et BC le nombre d'octets du bloc renuméroté).

Lignes 90 à 98

Déplacement de la partie du programme située après l'adresse contenue dans HL, de manière à ménager une place pour le bloc renuméroté.

Lignes 99 à 104

Insertion du bloc dans la place ainsi ménagée (avant le transfert répétitif de la ligne 104, HL pointe sur la première adresse de la place, BC est chargé avec le nombre d'octets du bloc renuméroté et DE pointe sur le début de la mémoire écran).

La ligne 105 récupère la valeur initiale de HL (qui est maintenant l'adresse de la première ligne du bloc renuméroté).

Lignes 106 à 126

Boucle de renumérotation des lignes du bloc *et* des numéros internes correspondants de tout le programme BASIC. Les numéros de ligne du bloc sont traités les uns après les autres par la boucle 108-126, et le retour de routine se fait après que le numéro L2 (celui de la dernière ligne du bloc) a été traité (RET Z de la ligne 119).

* *
*

Après ce morceau de bravoure, voici enfin le programme de chargement BASIC et de démonstration. Lancez le programme pour que la routine se charge, puis tapez CALL 40000, 540, 610, 511 suivi de RETURN (en mode direct, donc). Faites ensuite un listing d'écran et observez le résultat.

Signalons que la partie du programme allant des lignes 520 à 620 n'a pas de sens et n'est là que pour la démonstration.

```
 10 MODE 1:PRINT"CHARGEMENT EN COURS"
 20 '
 30 '
 40 MEMORY 39999
 50 ad=40000
 60 FOR i=1 TO 32
 70 v=0:READ a$,b$
 80 FOR j=1 TO LEN(a$)-1 STEP 2
 90 x=VAL("&"+MID$(a$,j,2)):v=v+x
```

```
100 POKE ad,x:ad=ad+1
110 NEXT j
120 IF v<>VAL("&"+b$) THEN PRINT"ERREUR
EN LIGNE ";160+(I-1)*10:END
130 NEXT i
140 '
150 '
160 DATA FE03C0D5216F0123CDAE9D2821,5AB
170 DATA 3E1DBE20F52B7E23FE202804FE,542
180 DATA 2C20EA361E235E2356EBD5CD4A,55B
190 DATA 9DE1722B7318D9DD6605DD6E04,616
200 DATA DD5603DD5E02CD0D9D2826EBF1,614
210 DATA E5C5F501000003C5CDA59D09C1,641
220 DATA E5ED52E120F3E1E509C1CDB89D,8CA
230 DATA C1E13007AF3E3F903007C13E07,4D2
240 DATA CD5ABBC9C5E51100C0EDB0EB2A,7D8
250 DATA 83AEED5223E5C1E1EBEDB01BDD,89A
260 DATA 4601DD4E00CD539DC5E1C1C5D5,730
270 DATA EBED5223E5C12A83AED1EBEDB8,8AF
280 DATA 23C1E51100C0EBEDB0E1DD4601,727
290 DATA DD4E00CD4A9DE5232323CDA09D,637
300 DATA CD739DDD6603DD6E02ED52E1C8,758
310 DATA C5CDA59D09C1C80318DFCD2C9D,6F6
320 DATA C8EBCD2C9DC8234E23462B2BE5,626
330 DATA ED52380809444DE13E023DC9E1,521
340 DATA AFC9E5216F01CDA59D2002E1C9,6C9
350 DATA E5232323CD989DE128030918EC,569
360 DATA 3E023DEBE1C9E52323235E2356,537
370 DATA E1C9D5216F01CD4A9DE5C5E1ED,83C
380 DATA 52E1380CC5CDA59DC5D1C12803,6CD
390 DATA 1918E9AFE5C1D1C9E5216F0123,6A2
400 DATA CDAE9D2002E1C93E1EBE20F32B,63C
410 DATA 7E23FE202804FE2C20E823CD98,5A5
420 DATA 9DCCA09D18DF7EBBC0237EBA2B,71C
430 DATA C97123702BC9234E23462B2B78,469
440 DATA B1C9EBE52A83AEED52E1EBC9E5,95E
450 DATA C5C5D1CD2C9D2015C1CD539DC5,769
460 DATA E1CDA59D200337E1C9CD4A9DE1,789
470 DATA ED52C9C1E1AFC9,522
480 '
490 '
500 '            DEMONSTRATION
510 '
```

```
520 END
530 REM ***
540 REM ***
550 X=1
560 ON X GOSUB 610,620
570 IF X=5 THEN 590 ELSE 580
580 REM ###
590 REM ###
600 END
610 RETURN
620 PRINT "ESSAI":IF X<>1 THEN GOTO 550
```

■ ADAPTATION AU 664 ET AU 6128

Sur ces deux machines, l'adresse de fin de programme BASIC est rangée en &AE66 au lieu de &AE83. Voici les trois lignes du programme de chargement qui devront être modifiées :

```
250 DATA 66AEED5223E5C1E1EBEDB01BDD,87D

270 DATA EBED5223E5C12A66AED1EBEDB8,892

440 DATA B1C9EBE52A66AEED52E1EBC9E5,941
```

16 RANGEMENT D'UN TABLEAU A UNE DIMENSION

L'utilisation de fichiers à accès direct n'étant pas possible sur le CPC, on est souvent amené à traiter les données par l'intermédiaire de tableaux. Le problème est que si ces tableaux sont importants, le temps de traitement par le BASIC peut être tout à fait considérable, voire rédhibitoire.

Pour y remédier, une seule solution : le langage machine. Nous vous proposons par conséquent, dans ce chapitre et dans le suivant, deux routines de travail sur les tableaux.

La première d'entre elles permet le rangement, par ordre croissant ou décroissant, d'un tableau entier à une dimension. Son format d'appel est le suivant :

CALL 40000, @T%(0)

- Le paramètre **@T%(0)** pointe sur la variable de rang 0 du tableau T% (il va sans dire que le nom de ce tableau peut être quelconque).

Nous allons commencer par voir la manière dont est rangé un tableau en RAM ; à titre d'exemple, voici comment se présente un tableau de nom TTT%, initialisé à 2 (''DIM TTT%(2)''), dont la première variable a été définie comme valant 1, la deuxième comme valant 2, et la troisième comme valant 3 (TTT%(0) = 1 ; TTT%(1) = 2 ; TTT%(2) = 3) :

Octet 1	84	code ASCII de T.
Octet 2	84	code ASCII de T.
Octet 3	212	code ASCII de T + &80 (signale la fin du nom).
Octet 4	1	type du tableau (entier ou réel).
Octet 5	9	nombre d'octets utilisés pour ranger les variables, plus 3 (octet faible).
Octet 6	0	idem, octet fort.
Octet 7	1	dimension du tableau (1 ou 2).
Octet 8	3	nombre de variables (octet faible).

Octet 9	0	idem, octet fort.
Octet 10	1	variable de rang 0 (octet faible).
Octet 11	0	idem, octet fort.
Octet 12	2	variable de rang 1 (octet faible).
Octet 13	0	idem, octet fort.
Octet 14	3	variable de rang 2 (octet faible).
Octet 15	0	idem, octet fort.

Il faut noter que la variable de rang 0 existe *toujours*, et que le nombre d'éléments du tableau est donc en fait supérieur de 1 à celui défini lors du dimensionnement. Bien souvent, ce rang 0 n'est pas utilisé, mais il est important de signaler que notre routine l'incluera dans le tri.

Si l'on se réfère à l'exemple de TTT%, précisons d'autre part que le paramètre @ TTT%(0) représenterait l'adresse de l'octet n° 10.

Le principe de rangement adopté est celui du *tri par bulles* qui, s'il n'est pas le plus rapide, a le mérite d'être simple.

Notre tableau étant constitué d'une série de variables, on pointe sur l'une d'entre elles et on la compare à la suivante. Si une inversion est nécessaire, elle est effectuée, puis le pointeur est repositionné au début du tableau et le processus reprend. Si le pointeur arrive en fin de tableau sans qu'aucune inversion ait été nécessaire, alors le programme est terminé.

Le nom de "tri par bulles" s'explique par une analogie : à chaque passe du pointeur, les éléments les plus "lourds" (ou les plus "légers", selon que le rangement est croissant ou décroissant) remontent progressivement vers le début du tableau.

Voici le listing assembleur de la routine :

```
1     40000  FE01        CP 1
2     40002  C0          RET NZ

3     40003  D5          PUSH DE
4     40004  E1          POP HL
5     40005  2B          DEC HL
6     40006  46          LD B,(HL)
7     40007  2B          DEC HL
8     40008  4E          LD C,(HL)
9     40009  0B          DEC BC
10    40010  23          INC HL
11    40011  23          INC HL
```

```
12    40012   13          INC DE
13    40013   13          INC DE
14    40014   E5          PUSH HL
15    40015   D5          PUSH DE
16    40016   C5          PUSH BC
17    40017   CD6F9C      CALL 40047      *
18    40020   FEFF        CP 255
19    40022   2008        JRNZ 40032
20    40024   CD939C      CALL 40083      *
21    40027   C1          POP BC
22    40028   D1          POP DE
23    40029   E1          POP HL
24    40030   18EE        JR 40014
25    40032   0B          DEC BC
26    40033   78          LD A,B
27    40034   B1          OR C
28    40035   2806        JRZ 40043
29    40037   23          INC HL
30    40038   23          INC HL
31    40039   13          INC DE
32    40040   13          INC DE
33    40041   18E6        JR 40017
34    40043   C1          POP BC
35    40044   C1          POP BC
36    40045   C1          POP BC
37    40046   C9          RET


38    40047   E5          PUSH HL
39    40048   D5          PUSH DE
40    40049   C5          PUSH BC
41    40050   4E          LD C,(HL)
42    40051   23          INC HL
43    40052   46          LD B,(HL)
44    40053   EB          EX HL,DE
45    40054   5E          LD E,(HL)
46    40055   23          INC HL
47    40056   56          LD D,(HL)
48    40057   C5          PUSH BC
49    40058   E1          POP HL
50    40059   7C          LD A,H
51    40060   AA          XOR D
52    40061   7C          LD A,H
```

```
53    40062   F2889C      JP P 40072      *
54    40065   87          ADD A,A
55    40066   9F          SBC A,A
56    40067   380A        JRC 40079
57    40069   3C          INC A
58    40070   1807        JR 40079
59    40072   BA          CP D
60    40073   20F7        JR NZ 40066
61    40075   7D          LD A,L
62    40076   93          SUB E
63    40077   20F3        JR NZ 40066
64    40079   C1          POP BC
65    40080   D1          POP DE
66    40081   E1          POP HL
67    40082   C9          RET

68    40083   4E          LD C,(HL)
69    40084   23          INC HL
70    40085   46          LD B,(HL)
71    40086   2B          DEC HL
72    40087   EB          EX HL,DE
73    40088   C5          PUSH BC
74    40089   EDA0        LDI
75    40091   EDA0        LDI
76    40093   C1          POP BC
77    40094   2B          DEC HL
78    40095   70          LD (HL),B
79    40096   2B          DEC HL
80    40097   71          LD (HL),C
81    40098   C9          RET
```

Voyons d'abord les deux sous-programmes :

■ SOUS-PROGRAMME 1 (LIGNES 38 A 67)

Ce sous-programme est chargé de comparer le nombre pointé par HL (que nous appellerons N1) et le nombre pointé par DE (que nous appellerons N2). Tous les registres sont préservés en sortie, sauf A dont la valeur dépend du résultat de la comparaison :

Si (DE) > (HL), alors A = &FF.

Si (DE) < (HL), alors A=&01.
Si (DE) = (HL), alors A=&00.

Les registres sont tout d'abord préservés (lignes 38 à 40), après quoi N1 est chargé dans BC (lignes 41 à 43). Le registre HL est ensuite pointé sur N2 (ligne 44) qui est chargé dans DE (lignes 45 à 47). Le nombre N1 passe finalement dans HL (lignes 48 et 49), puis le test est réalisé par les lignes 52 à 63 (il ne présente pas de difficulté particulière et sa compréhension ne demande qu'une lecture attentive).

Les valeurs initiales des registres HL, DE et BC sont récupérées aux lignes 64 à 66, avant que le retour soit effectué.

■ SOUS-PROGRAMME 2 (LIGNES 68 A 81)

Il est chargé d'inverser le nombre pointé par HL (N1) et celui pointé par DE (N2) ; tous les registres sont modifiés en sortie.

Le nombre N1 est chargé dans BC (lignes 68 à 70), puis le contenu des registres HL et DE est échangé en ligne 72 (HL pointe donc sur N2 et DE sur N1). Les lignes 74 et 75 transfèrent N2 en lieu et place de N1 (notez que BC est affecté par les instructions de transfert et doit par conséquent être préservé), puis le contenu de BC (=N1) est chargé dans l'ex-placard de N2 (lignes 78 à 80).

Le programme principal commence par le chargement de HL avec le paramètre @T%(0) (lignes 3 et 4) ; ce registre pointe donc maintenant sur l'octet faible de la variable de rang 0. Le registre BC est chargé avec le nombre d'éléments du tableau moins 1 (lignes 5 à 9) et, après les quatre décrémentations des lignes 10 à 13, HL pointe sur la variable de rang 0 et DE sur la variable de rang 1. La boucle de rangement peut alors commencer.

Les trois registres sont d'abord préservés sur la pile (il ne faut pas oublier en effet que si une inversion a lieu le processus devra être repris depuis le début. Il suffira dans ce cas de récupérer ces valeurs initiales).

La comparaison entre le nombre pointé par HL (que l'on appellera NX) et celui pointé par DE (que l'on appellera NX+1) est effectuée en ligne 17 grâce au sous-programme 1. Deux cas sont alors possibles :

- Si A = &FF au retour, donc si NX+1 > NX, le programme passe en ligne 20 où l'inversion des deux nombres est effectuée. Les trois registres sont ensuite restaurés et la boucle reprend depuis le départ.
- Si A est différent de &FF, aucune inversion n'est nécessaire et le programme passe en ligne 25, où le compteur BC est décrémenté puis testé. Lorsque la valeur 0 est atteinte, c'est donc que tout le tableau a été parcouru ; dans ce cas, le saut de la ligne 28 envoie vers le nettoyage de la pile et le retour de sous-programme.
  Si BC est plus grand que 0, les registres HL et DE sont respectivement pointés sur NX+1 et NX+2 (lignes 29 à 32) et la boucle reprend cette fois en ligne 17.

Voici maintenant le programme de chargement BASIC (cette routine fonctionne indifféremment sur les trois machines). Lancez-le d'abord tel quel, puis essayez en inversant le sens de tri (il suffit bien entendu pour cela de mettre à 1 l'octet 40021).

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 7
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE01C0D5E12B462B4E0B232313,4C3
170 DATA 13E5D5C5CD6F9CFEFF2008CD93,7EF
180 DATA 9CC1D1E118EE0B78B128062323,5BD
190 DATA 131318E6C1C1C1C9E5D5C54E23,720
200 DATA 46EB5E2356C5E17CAA7CF2889C,766
210 DATA 879F380A3C1807BA20F77D9320,4C4
220 DATA F3C1D1E1C94E23462BEBC5EDA0,84E
230 DATA EDA0C12B702B71C9,44E
```

```
240 '
250 '
260 '          DEMONSTRATION
270 '
280 CLS:DIM T%(23):FOR I=0 TO 23
290 T%(I)=INT(RND*500):NEXT I
300 LOCATE 1,12:PRINT"AVANT TRI:"
310 FOR I=0 TO 23:LOCATE 12,I+1
320 PRINT T%(I):NEXT
330 '
340 CALL 40000, @ T%(0):PEN 3
350 LOCATE 20,12:PRINT"APRES TRI:"
360 FOR I=0 TO 23:LOCATE 32,I+1
370 PRINT T%(I):NEXT
380 PEN 1:GOTO 380
```

17 RANGEMENT DANS UN TABLEAU BIDIMENSIONNÉ

Après les tableaux à une dimension, nous allons en toute bonne logique nous attaquer aux tableaux à deux dimensions. C'est ainsi que la routine proposée dans ce chapitre a pour fonction de ranger la colonne C d'un tableau bidimensionné.

Pour fixer les idées, précisons que nous parlerons toujours de colonnes et de lignes, ces dernières étant représentées par le second chiffre (il va sans dire que ce choix, tout à fait arbitraire, ne recouvre en tout état de cause qu'une vue de l'esprit).

L'instruction DIM TTT%(1,2), par exemple, créera un tableau de deux colonnes et trois lignes (il ne faut pas oublier la ligne et la colonne 0).

Le format d'appel de la routine est le suivant :

```
CALL 40000, @ T%(0,0),C
```

- Le paramètre **C** désigne la colonne qui doit être rangée, et **@ T%(0,0)** représente l'adresse de la variable située à l'intersection de la colonne 0 et de la ligne 0.

L'organisation en RAM d'un tableau à deux dimensions diffère quelque peu de celle d'un tableau simple. Voyons par exemple comment se présenterait un tableau dimensionné par DIM TTT%(1,2) et dont la ligne 0 serait remplie de 1, la ligne 1 remplie de 2, et la ligne 2 remplie de 3 :

Octet 1	84	code ASCII de 'T'.
Octet 2	84	idem.
Octet 3	212	code ASCII de 'T'+&80 (signale la fin du nom).
Octet 4	1	type du tableau (entier ou réel).
Octet 5	17	nombre d'octets (poids faible).
Octet 6	0	idem, poids fort.
Octet 7	2	indique un tableau à deux dimensions.

Octet 8	3	nombre de lignes (poids faible).
Octet 9	0	idem, poids fort.
Octet 10	2	nombre de colonnes (poids faible).
Octet 11	0	idem, poids fort.
Octet 12	1	variable de rang (0,0), poids faible.
Octet 13	0	idem, poids fort.
Octet 14	1	variable de rang (1,0), poids faible.
Octet 15	0	idem, poids fort.
Octet 16	2	variable de rang (0,1), poids faible.
Octet 17	0	idem, poids fort.
Octet 18	2	variable de rang (1,1), poids faible.
Octet 19	0	idem, poids fort.
Octet 20	3	variable de rang (0,2), poids faible.
Octet 21	0	idem, poids fort.
Octet 22	3	variable de rang (1,2), poids faible.
Octet 23	0	idem, poids fort.

Notons que, dans ce cas précis, le paramètre @ TTT%(0,0) représenterait l'adresse de l'octet 12.

Comme on le voit, le tableau est rangé ligne après ligne (octets 12 à 15 : colonnes 0 et 1 de la première ligne ; octets 16 à 19 : colonnes 0 et 1 de la deuxième ligne ; octets 20 à 23 : colonnes 0 et 1 de la troisième ligne). Il sera nécessaire de se souvenir de cette organisation particulière pour la bonne compréhension du programme.

Voici le listing assembleur :

```
1     40000  FE02       CP 2
2     40002  C0         RET NZ

3     40003  DD6603     LD H,(IX+3)
4     40006  DD6E02     LD L,(IX+2)
5     40009  E5         PUSH HL
6     40010  D5         PUSH DE
7     40011  2B         DEC HL
8     40012  56         LD D,(HL)
9     40013  2B         DEC HL
10    40014  5E         LD E,(HL)
11    40015  ED53CA9C   LD (40138),DE *
```

```
12    40019  2B          DEC HL
13    40020  46          LD B,(HL)
14    40021  2B          DEC HL
15    40022  4E          LD C,(HL)
16    40023  B           DEC BC
17    40024  E1          POP HL
18    40025  29          ADD HL,HL
19    40026  D1          POP DE
20    40027  19          ADD HL,DE
21    40028  EB          EX HL,DE
22    40029  2ACA9C      LD HL,(40138) *
23    40032  29          ADD HL,HL
24    40033  19          ADD HL,DE
25    40034  EB          EX HL,DE

26    40035  E5          PUSH HL
27    40036  D5          PUSH DE
28    40037  C5          PUSH BC
29    40038  CD839C      CALL 40067    *
30    40041  FEFF        CP 255
31    40043  2008        JRNZ 40053
32    40045  CDA79C      CALL 40103    *
33    40048  C1          POP BC
34    40049  D1          POP DE
35    40050  E1          POP HL
36    40051  18EE        JR 40035
37    40053  0B          DEC BC
38    40054  78          LD A,B
39    40055  B1          OR C
40    40056  2805        JRZ 40063
41    40058  CDB79C      CALL 40119    *
42    40061  18E7        JR 40038
43    40063  C1          POP BC
44    40064  C1          POP BC
45    40065  C1          POP BC
46    40066  C9          RET


47    40067  E5          PUSH HL
48    40068  D5          PUSH DE
49    40069  C5          PUSH BC
50    40070  4E          LD C,(HL)
51    40071  23          INC HL
```

```
52   40072  46        LD B,(HL)
53   40073  EB        EX HL,DE
54   40074  5E        LD E,(HL)
55   40075  23        INC HL
56   40076  56        LD D,(HL)
57   40077  C5        PUSH BC
58   40078  E1        POP HL
59   40079  7C        LD A,H
60   40080  AA        XOR D
61   40081  7C        LD A,H
62   40082  F29C9C    JP P 40092     *
63   40085  87        ADD A,A
64   40086  9F        SBC A,A
65   40087  380A      JRC 40099
66   40089  3C        INC A
67   40090  1807      JR 40099
68   40092  BA        CP D
69   40093  20F7      JR NZ 40086
70   40095  7D        LD A,L
71   40096  93        SUB E
72   40097  20F3      JR NZ 40086
73   40099  C1        POP BC
74   40100  D1        POP DE
75   40101  E1        POP HL
76   40102  C9        RET

77   40103  4E        LD C,(HL)
78   40104  23        INC HL
79   40105  46        LD B,(HL)
80   40106  2B        DEC HL
81   40107  EB        EX HL,DE
82   40108  C5        PUSH BC
83   40109  EDA0      LDI
84   40111  EDA0      LDI
85   40113  C1        POP BC
86   40114  2B        DEC HL
87   40115  70        LD (HL),B
88   40116  2B        DEC HL
89   40117  71        LD (HL),C
90   40118  C9        RET

91   40119  C5        PUSH BC
92   40120  E5        PUSH HL
```

```
93    40121   2ACA9C      LD HL,(40138) *
94    40124   29          ADD HL,HL
95    40125   C1          POP BC
96    40126   09          ADD HL,BC
97    40127   EB          EX HL,DE
98    40128   E5          PUSH HL
99    40129   2ACA9C      LD HL,(40138) *
100   40132   29          ADD HL,HL
101   40133   C1          POP BC
102   40134   09          ADD HL,BC
103   40135   EB          EX HL,DE
104   40136   C1          POP BC
105   40137   C9          RET


106   40138   0000        Placard
```

Trois sous-programmes sont mis en place, dont les deux premiers sont en tout point identiques à ceux du chapitre précédent (la méthode de rangement utilisée est la même). Voyons donc le troisième :

■ SOUS-PROGRAMME 3 (LIGNES 91 A 105)

Ce sous-programme est appelé pour faire passer le pointage des registres HL et DE d'une colonne à une autre de même numéro, mais sur la ligne suivante.

En se référant à l'exemple précédent, imaginons que HL pointe sur l'octet 12 et DE sur l'octet 16 (donc respectivement sur la colonne 1, ligne 1 et sur la colonne 1, ligne 2) ; l'appel de ce sous-programme aurait pour effet de pointer HL sur l'octet 16, et DE sur l'octet 20 (donc respectivement sur la colonne 1, ligne 2 et sur la colonne 1, ligne 3). Cela servira bien sûr à comparer successivement entre elles toutes les variables d'une même colonne.

Sachant que chaque variable est stockée sur 2 octets, il est facile d'en déduire que le passage d'une colonne donnée à la même colonne mais sur la ligne suivante sera obtenu simplement en ajoutant le nombre de colonnes×2 aux contenus initiaux des registres (vous pouvez le vérifier sur le tableau TTT% ; pour passer de l'octet 12 à l'octet 16, par exemple, il faut ajouter 2×2).

Les deux registres sont tout d'abord préservés, puis HL est chargé avec le contenu du placard d'adresse 40138, dans lequel a été rangé, au départ de la routine, le nombre de colonnes du tableau. La ligne 94 multiplie ce nombre par 2, puis le contenu initial de HL passe dans BC (ligne 95) et l'addition est effectuée (ligne 96).

Le résultat ainsi trouvé est préservé dans le registre DE, en même temps que HL est chargé avec le contenu initial de DE (ligne 97). Le même processus a alors lieu (lignes 98 à 102), à la fin duquel HL contient la valeur qui devra être chargée dans DE. Les deux registres sont échangés (ligne 103), BC est restauré et le retour effectué.

Le programme principal commence par charger HL avec l'adresse de la variable de rang (0,0), puis les deux paramètres sont préservés sur la pile.

Le nombre de colonnes du tableau est ensuite rangé dans le placard d'adresse 40138 (lignes 7 à 11) et le nombre de lignes moins 1 dans BC (lignes 12 à 16), qui servira bien sûr de compteur.

Les lignes 17 à 20 calculent la position de HL au départ de la boucle, qui doit être l'adresse de la variable de rang (C,0) ; cette adresse est égale à @ TTT%(0,0) + (C × 2).

Les lignes 22 à 24 calculent l'adresse de DE au départ de la boucle ; cette adresse doit être égale à l'adresse de départ de HL + (nombre de colonnes × 2).

Un exemple, pour que tout cela soit bien clair : si nous voulions ranger la colonne 1 du tableau TTT%, HL devrait pointer au départ sur l'octet 14, et DE devrait pointer sur l'octet 18.

La boucle de comparaison et de rangement (lignes 26 à 46) est pratiquement la même que celle du chapitre précédent. Seule la ligne 41 (appel du sous-programme "avançant" HL et DE d'une ligne) est différente ; nous ne reviendrons donc pas sur cette partie du programme.

Voici le programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 10
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
```

```
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA FE02C0DD6603DD6E02E5D52B56,68E
170 DATA 2B5EED53CA9C2B462B4E0BE129,52E
180 DATA D119EB2ACA9C2919EBE5D5C5CD,7DE
190 DATA 839CFEFF2008CDA79CC1D1E118,7DF
200 DATA EE0B78B12805CDB79C18E7C1C1,6F0
210 DATA C1C9E5D5C54E2346EB5E2356C5,747
220 DATA E17CAA7CF29C9C879F380A3C18,669
230 DATA 07BA20F77D9320F3C1D1E1C94E,785
240 DATA 23462BEBC5EDA0EDA0C12B702B,6E5
250 DATA 71C9C5E52ACA9C29C109EBE52A,761
260 DATA CA9C29C109EBC1C90000,4CE
270 '
280 '
290 '          DEMONSTRATION
300 '
310 DIM T%(3,23)
320 FOR I=0 TO 3:FOR J=0 TO 23
330 X=INT(RND*1000):T%(I,J)=X
340 NEXT J,I:CLS
350 LOCATE 1,12:PRINT"AVANT TRI:"
360 LOCATE 22,12:PRINT"APRES TRI:"
370 '
380 FOR I=0 TO 23:LOCATE 12,I+1
390 PRINT T%(2,I):NEXT
400 CALL 40000, @ T%(0,0),2
410 FOR I=0 TO 23:LOCATE 32,I+1
420 PRINT T%(2,I):NEXT
430 GOTO 430
```

18, 19, 20 ÉCRITURE EN MÉMOIRE ÉCRAN

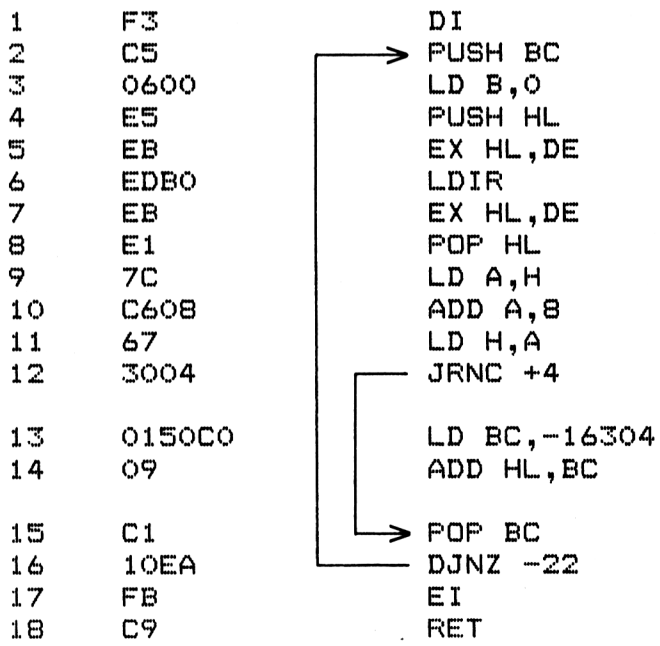
Les trois programmes qui vont être présentés ci-après sont un peu particuliers. Il ne s'agit pas, en effet, de routines destinées à être appelées à partir du BASIC, mais bien de sous-programmes devant être gérés à partir d'un programme principal en langage machine.

Deux d'entre eux permettent l'écriture directe en mémoire écran d'une zone de mémoire RAM (en mode normal ou transparent), et le troisième l'effaçage d'une zone de mémoire écran.

Ces trois sous-programmes offrent des perspectives extrêmement intéressantes autant que multiples, en particulier pour tout ce qui concerne les animations d'écran. Encore faut-il, naturellement, les utiliser à bon escient. Vous trouverez plus loin trois exemples d'application, dont un jeu d'arcade présentant le déplacement simultané de cinq mobiles.

Là encore, il est nécessaire de bien connaître l'organisation de la mémoire écran (revoir éventuellement à ce sujet le Programme 12).

Voici le listing assembleur du premier sous-programme, qui dessine une image en mémoire écran à partir d'une image en RAM, cela en écriture normale :



Les registres (qui sont tous modifiés en sortie) doivent être, avant l'appel, chargés de la manière suivante :

- HL doit pointer sur l'adresse écran à partir de laquelle doit être écrite l'image (cette adresse représentera l'octet haut/gauche de l'image).
- DE doit pointer sur le début de l'image en RAM.
- B doit contenir le nombre de traits de l'image.
- C doit contenir le nombre de colonnes de l'image.

Le nombre de colonnes et de traits s'entendent en terme d'octets et n'ont bien sûr rien à voir avec les colonnes et lignes du mode texte. (Si par exemple vous voulez dessiner, en haut et à gauche de l'écran, un pavé rouge de 3 octets de large sur 9 de haut, BC devra être chargé avec la valeur &0903 ; HL devra être chargé avec &C000, et DE devra pointer sur une zone quelconque de 27 octets, chacun contenant la valeur &FF.)

L'image stockée en RAM doit être organisée trait après trait. Si vous vouliez afficher, au lieu du pavé rouge évoqué plus haut, un pavé rayé horizontalement en rouge et en bleu, son image devrait être, dans l'ordre : &FF, &FF, &FF, &0F, &0F, &0F, &FF, &FF, &FF, etc.

Voyons maintenant le fonctionnement du sous-programme :

L'instruction DI commence d'abord par interdire toutes les interruptions internes masquables (c'est-à-dire celles qui peuvent être interdites). Cela est surtout utile quand le sous-programme est utilisé pour des animations d'écran. Il ne faut pas oublier qu'entre deux balayages d'écran par le rayon il ne se passe qu'un cinquantième de seconde. Un programme d'animation ne doit donc pas dépasser ce délai si l'on veut éviter des mouvements saccadés ou des effets optiques désagréables. Par conséquent, toute perte de temps est à proscrire... et en particulier celles qui, comme les interruptions internes, ne concernent pas directement le programme.

Après sauvegarde du registre BC, B est mis à 0 en prévision du transfert répétitif de la ligne 7 qui écrit un trait de l'image (trait composé de x octets, x étant le nombre de colonnes, chargé dans C au départ et servant de compteur au transfert répétitif).

A la fin de l'écriture de chaque trait, DE se retrouve pointé sur le début de l'image en RAM du trait suivant. HL est ensuite préparé pour dessiner ce dernier : les lignes 9, 10 et 11 ont pour effet de lui ajouter 2048 ($2048 = 256 \times 8$ et c'est pourquoi il suffit d'ajouter 8 au registre de poids fort). Si cette opération ne génère pas de report, donc

si le 8e trait n'est pas dépassé, le saut à la ligne 15 s'effectue. Là, BC est récupéré puis la ligne 16 teste si tous les traits ont été dessinés (c'est cette fois le registre B qui sert de compteur). Si c'est le cas, les interruptions sont de nouveau autorisées (EI) et le sous-programme se termine. Dans le cas contraire, la boucle reprend en ligne 3.

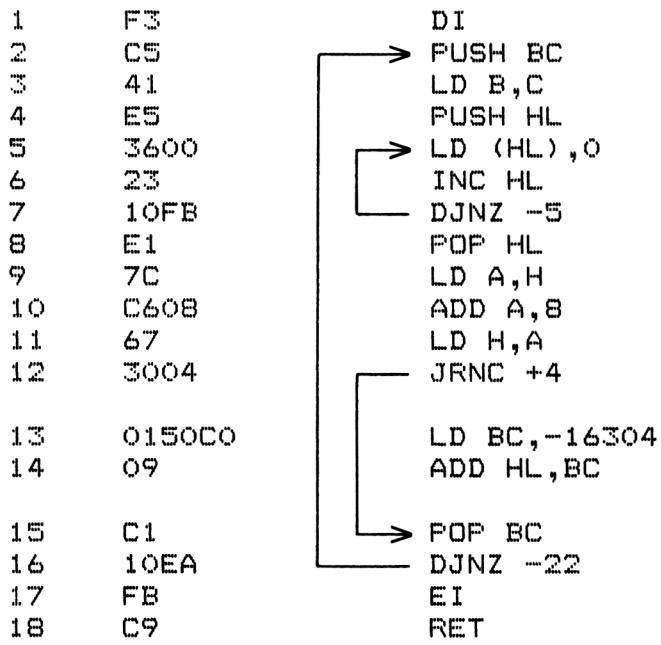
Si un report a été généré en ligne 10, c'est que le registre HL a dépassé la valeur maximale de 65535. Dans ce cas, une correction est apportée par les lignes 13 et 14.

Voici par exemple les adresses des dix premiers octets du bord gauche et en partant du haut de la mémoire écran :

49152
51200
53248
55296
57344
59392
61440
63488
49232
51280

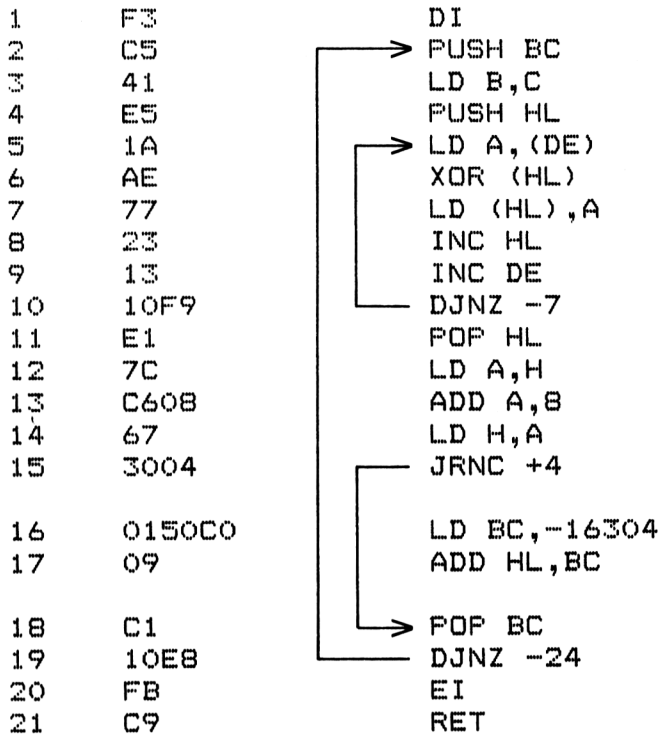
En partant de 49152 et en ajoutant chaque fois 2048, aucun problème pour arriver à 63488. L'addition suivante, par contre, aura comme résultat 65536, c'est-à-dire plus que la valeur maximale que l'on peut charger dans un registre double (65535). Cela provoquera un report que l'on compensera en enlevant 16304 au résultat illégal (65536 – 16304 = 49232).

Voici maintenant le listing du deuxième sous-programme, complémentaire du précédent puisqu'il permet l'effaçage d'une zone d'écran.



Son fonctionnement est presque identique, et les registres HL, B et C doivent être chargés de la même manière. Cette fois, pourtant, il ne s'agit pas de recopier une zone RAM, mais simplement de mettre à 0 toutes les adresses concernées. Le registre DE n'est par conséquent pas utilisé et le transfert répétitif est remplacé par la boucle constituée des lignes 5, 6 et 7.

Nous avons parlé d'écriture normale, en opposition avec l'écriture transparente. Ces deux sous-programmes, en effet, écrivent ou effacent en mémoire écran sans se préoccuper de son contenu initial. Si une image est écrite sur une autre, cette dernière est définitivement perdue, ce qui peut être parfois gênant. Nous vous proposons donc un troisième sous-programme permettant l'écriture en mode transparent et donc capable de restituer la mémoire écran dans son intégralité.



Les registres doivent être chargés comme pour le premier sous-programme. Ils sont tous modifiés en sortie. Là encore, le principe ne diffère guère de celui des programmes précédents. Seule la boucle d'écriture d'un trait (lignes 5 à 10) présente une particularité : l'octet-image qui doit être écrit en mémoire écran est tout d'abord chargé dans le registre A (ligne 5). Puis un OU exclusif logique est réalisé, *avant l'écriture, entre ce registre et la valeur pointée par HL en mémoire écran*. Pour retrouver le contenu initial de l'écran, il suffira ensuite de répéter exactement la même opération.

Exemple

Imaginons que la lettre A soit affichée en haut et à gauche de l'écran et que, pour des raisons connues de vous seul, vous souhaitiez afficher à la même place et provisoirement un pavé rouge de 2 octets de large sur 8 de haut.

HL devra être chargé avec 49152, BC avec &0802, et DE devra pointer sur le début d'une zone RAM contenant 16 fois la valeur &FF.

Un premier appel de la routine affichera votre pavé sans détruire la lettre A. Un second appel (les registres étant rechargés pareillement) restituera la lettre en l'état.

Voici l'illustration de ce principe grâce à un programme BASIC de quelques lignes. Observez ce qui se passe chaque fois que vous appuyez sur une touche...

```
10 MODE 1:PRINT "A":GOSUB 100
20 FOR i=49152 TO 63488 STEP 2048
30 FOR j=i TO i+1
40 x=255 XOR PEEK(j):POKE j,x
50 NEXT j,i
60 GOSUB 100
70 GOTO 20
100 rep$=INKEY$:IF rep$="" THEN 100 ELSE
 RETURN
```

Comme promis, voici maintenant deux exemples d'application de ces sous-programmes.

Le premier dessine une trame sur l'écran à partir d'une image de 16 octets écrite en RAM, des adresses 42000 à 42015. Cette image aura, sur l'écran, 2 octets de large sur 8 de haut et sera réécrite jusqu'au remplissage complet de l'écran.

Voici tout d'abord le listing assembleur :

```
1     40000  010208     LD BC,&0802
2     40003  1110A4     LD DE,42000    *
3     40006  2100C0     LD HL,49152
4     40009  C5         PUSH BC
5     40010  D5         PUSH DE
6     40011  E5         PUSH HL
7     40012  CD5A9C     CALL 40026     *
8     40015  E1         POP HL
9     40016  D1         POP DE
10    40017  C1         POP BC
11    40018  23         INC HL
12    40019  23         INC HL
13    40020  3EC8       LD A,&C8
14    40022  BC         CP H
15    40023  20F0       JRNZ 40009
```

```
16    40025  C9          RET


17    40026  F3          DI
18    40027  C5          PUSH BC
19    40028  0600        LD B,0
20    40030  E5          PUSH HL
21    40031  EB          EX HL,DE
22    40032  EDB0        LDIR
23    40034  EB          EX HL,DE
24    40035  E1          POP HL
25    40036  7C          LD A,H
26    40037  C608        ADD A,8
27    40039  67          LD H,A
28    40040  3004        JRNC 40046
29    40042  0150C0      LD BC,-16304
30    40045  09          ADD HL,BC
31    40046  C1          POP BC
32    40047  10EA        DJNZ 40027
33    40049  FB          EI
34    40050  C9          RET
```

Le sous-programme d'écriture normale est implanté à partir de l'adresse 40026. Chaque fois que l'écriture d'une image est terminée, HL est augmenté de 2 (lignes 11 et 12), et pointe donc sur l'adresse suivante.

Le programme ne présente pas de particularité, si ce n'est la vérification effectuée par les lignes 13, 14 et 15 : si HL atteint la valeur 51200, donc si son octet fort atteint &C8, c'est que tout l'écran est rempli et que le programme doit se terminer. Sinon, la boucle reprend en ligne 4.

Voici ensuite le listing de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 3
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
```

```
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA 0102081110A42100C0C5D5E5CD,4FD
170 DATA 5A9CE1D1C123233EC8BC20F0C9,74A
180 DATA F3C50600E5EBEDB0EBE17CC608,841
190 DATA 6730040150C009C110EAFBC9,534
200 '
210 '
220 FOR I=42000 TO 42015:READ X$
230 POKE I,VAL("&"+X$):NEXT
240 DATA 19,89,26,46,46,26,89,19
250 DATA 89,19,46,26,26,46,19,89
260 '
270 MODE 1:CALL 40000
280 GOTO 280
```

La préparation de l'image en RAM est réalisée par les lignes 220 à 250. Vous pouvez bien entendu modifier à votre guise les données des lignes DATA.

*
* *

Le second exemple utilise le sous-programme d'écriture transparente et montre la qualité des déplacements que l'on peut obtenir.

Le programme boucle sans arrêt et ne rend la main au BASIC que si l'on presse la barre d'espace.

Voici le listing assembleur :

```
1     40000   CD19BD     CALL &BD19

2     40003   2A8B9C     LD HL,(40075) *
3     40006   1110A4     LD DE,42000   *
4     40009   010410     LD BC,&1004
5     40012   CD709C     CALL 40048    *
```

```
6     40015   2A8B9C      LD HL,(40075) *
7     40018   010008      LD BC,2048
8     40021   09          ADD HL,BC
9     40022   3004        JRNC 40028
10    40024   0150C0      LD BC,-16304
11    40027   09          ADD HL,BC
12    40028   228B9C      LD (40075),HL *
13    40031   1110A4      LD DE,42000   *
14    40034   010410      LD BC,&1004
15    40037   CD709C      CALL 40048    *

16    40040   3E2F        LD A,&2F
17    40042   CD1EBB      CALL &BB1E
18    40045   28D1        JRZ 40000
19    40047   C9          RET

20    40048   0000        Placard


21    40050   F3          DI
22    40051   C5          PUSH BC
23    40052   41          LD B,C
24    40053   E5          PUSH HL
25    40054   1A          LD A,(DE)
26    40055   AE          XOR (HL)
27    40056   77          LD (HL),A
28    40057   23          INC HL
29    40058   13          INC DE
30    40059   10F9        DJNZ 40054
31    40061   E1          POP HL
32    40062   7C          LD A,H
33    40063   C608        ADD A,8
34    40065   67          LD H,A
35    40066   3004        JRNC 40072
36    40068   0150C0      LD BC,-16304
37    40071   09          ADD HL,BC
38    40072   C1          POP BC
39    40073   10E8        DJNZ 40051
40    40075   FB          EI
41    40076   C9          RET
```

Le sous-programme est implanté en 40050, et l'image de l'objet (un simple pavé rouge de 4 octets de large et de 16 de haut) des adres-

ses 42000 à 42063. Le placard d'adresse 40048 est prévu pour le rangement des adresses successives du mobile.

Comme pour tout programme d'animation, celui-ci commence par appeler la routine &BD19, qui le synchronise avec le balayage de l'écran. Le registre HL est chargé, en ligne 2, avec l'adresse actuelle du mobile, DE avec l'adresse de son image, B avec &10 (= 16 en décimal), et C avec &4.

Le mobile (qui doit être affiché sur l'écran *avant* que ce programme ne démarre) est effacé en ligne 5. Puis la nouvelle adresse où va être dessiné le mobile est calculée par les lignes 6, 7 et 8. (La valeur 2048, choisie arbitrairement, est additionnée à l'ancienne adresse. Cela aura donc pour effet de faire descendre le mobile d'un octet par rapport à sa précédente situation.) Il faut noter qu'un dépassement éventuel est vérifié (lignes 9, 10 et 11) selon le principe déjà exposé plus haut.

Une remarque, à ce propos : cette vérification n'est valable que pour des déplacements vers le bas. Si par exemple l'objet devait être déplacé vers le haut (en soustrayant par exemple 2048 au lieu de l'ajouter), il faudrait alors vérifier que la nouvelle adresse n'est en aucun cas inférieure à 49152. Ce qui n'est pas bien difficile, puisqu'il suffit tout simplement de vérifier que l'octet fort reste supérieur ou égal à &C0.

Les lignes 16, 17 et 18 testent si la barre d'espace est pressée, cela grâce à l'appel de la routine &BB1E. Le registre A doit préalablement être chargé avec le code de la touche qui doit être testée (&2F pour la barre d'espace). Au retour de la routine, l'indicateur de 0 n'est vrai que si la touche n'est pas pressée. Sinon, NZ est vrai.

Voici le listing de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 5
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
```

```
130 NEXT I
140 '
150 '
160 DATA CD19BD2A8B9C1110A4010410CD,49B
170 DATA 709C2A8B9C0100080930040150,2F4
180 DATA C009228B9C1110A4010410CD70,429
190 DATA 9C3E2FCD1EBB28D1C9F3C541E5,74F
200 DATA 1AAE77231310F9E17CC6086730,540
210 DATA 040150C009C110E8FBC90000,49B
220 '
230 '
240 FOR I=42000 TO 42063
250 POKE I,255:NEXT
260 '
270 GOSUB 430:CALL 40000
280 '
290 POKE 40019,1:POKE 40020,0
300 GOSUB 430:CALL 40000
310 '
320 POKE 40019,1:POKE 40020,8
330 GOSUB 430:CALL 40000
340 '
350 POKE 40019,0:POKE 40020,&20
360 GOSUB 430:CALL 40000
370 '
380 POKE 40019,&FE:POKE 40020,&7
390 GOSUB 430:CALL 40000
400 PEN 1:END
410 '
420 '
430 POKE 40075,0:POKE 40076,&C0
440 MODE 1:PEN 3
450 PRINT CHR$(143)+CHR$(143)
460 PRINT CHR$(143)+CHR$(143):PEN 1
470 LOCATE 1,12:PRINT STRING$(40,"*")
480 RETURN
```

L'image du pavé est implantée en mémoire par les lignes 240 et 250. Le sous-programme 430 exécute plusieurs tâches :

- Il charge le placard avec l'adresse de départ du mobile (en l'occurrence le coin haut/gauche de l'écran). Il va de soi que cette adresse peut être choisie librement.

- Il dessine le mobile à cette adresse avant l'appel du programme en langage machine. Plutôt que de vous fournir une explication brumeuse sur la nécessité de dessiner le mobile au préalable, nous vous conseillons d'essayer le programme en supprimant les lignes 450 et 460, de voir ce que cela entraîne et d'y réfléchir.

Le programme propose cinq types de déplacements qui auront lieu successivement chaque fois que vous appuierez sur la barre d'espace. Vous pouvez en effet constater qu'à chaque retour au BASIC, et avant tout nouvel appel en 40000, les lignes 290, 320, 350 et 380 modifient les emplacements mémoire 40019 et 40020, c'est-à-dire la valeur chargée dans BC avant qu'il soit additionné à l'ancienne adresse du mobile (ligne 7 du listing assembleur).

Vous pouvez vous amuser à modifier cette valeur et vous exclamer devant les résultats, mais attention aux valeurs négatives qui feraient monter le mobile. Il faudrait dans ce cas installer une protection pour qu'il n'aille pas se promener vers des adresses inférieures à 49152 (revoir à ce sujet le paragraphe du Programme 12 traitant de la mémoire écran).

21 JEU D'ARCADE

Voici enfin, pour terminer sur une note ludique, le jeu d'arcade que nous vous avions promis. Comme il a été dit, les trois sous-programmes du chapitre précédent sont utilisés et ils permettent de gérer les déplacements simultanés de cinq mobiles.

La règle du jeu est des plus simples :

Deux fusées se déplacent sur l'écran. Vous devez les détruire en envoyant des missiles avec un canon. Vous pouvez diriger le canon grâce aux touches curseur (droite ou gauche), et tirer grâce à la touche COPY (notons qu'il est possible de tirer tout en se déplaçant ; il n'est par contre pas possible de tirer un nouveau missile tant que le précédent n'a pas atteint son but ou n'est pas sorti de l'écran).

Attention ! Si l'astéroïde mobile qui parcourt sans arrêt l'écran touche le canon, celui-ci est détruit, le jeu s'arrête et votre score s'affiche (rien ne vous empêche, si vous êtes habile, de détruire l'astéroïde à coups de missiles).

Attention également aux astéroïdes fixes qui se trouveraient sur la trajectoire de vos projectiles. Dans ce cas tout explose, missiles aussi bien qu'astéroïdes, mais cela ne rapporte aucun point.

Vous disposez au départ de 20 missiles, et chaque coup au but rapporte 1 point. Si vous réussissez à détruire *les deux fusées* avant qu'elles n'atteignent la fin de leur parcours et qu'elles ne soient réinitialisées, un bonus de 1 point vous est accordé. Lorsque les vingt missiles ont été tirés, le jeu s'arrête et le score s'affiche (pour information, le score maximal est de 30 points).

Important

Une ligne du programme de chargement BASIC que vous trouverez plus loin doit être modifiée pour le 664 et le 6128 :

```
170 DATA 002135B6AFBE2320113E02BE20,3EB
```

Listing assembleur :

```
        1) Initialisation

1       40000   DD2118A4    LD IX,42008     *
```

```
2     40004    FD21719D    LD IY,40305     *
3     40008    CD19BD      CALL &BD19
4     40011    00
5     40012    00
6     40013    00
```

2) Interrogation clavier

```
7     40014    21EBB4      LD HL,&B4EB     #
8     40017    AF          XOR A
9     40018    BE          CP (HL)
10    40019    23          INC HL
11    40020    2011        JRNZ 40039
12    40022    3E02        LD A,2
13    40024    BE          CP (HL)
14    40025    200C        JRNZ 40039
15    40027    AF          XOR A
16    40028    FDBEB6      CP (IY-74)
17    40031    2806        JRZ 40039
18    40033    CDB59E      CALL 40629      *
19    40036    C3F49C      JP 40180        *
20    40039    3E03        LD A,3
21    40041    BE          CP (HL)
22    40042    284A        JRZ 40118
23    40044    3E01        LD A,1
24    40046    BE          CP (HL)
25    40047    284E        JRZ 40127
26    40049    2B          DEC HL
27    40050    BE          CP (HL)
28    40051    C8          RET Z
29    40052    3C          INC A
30    40053    BE          CP (HL)
31    40054    2803        JRZ 40059
32    40056    C3F49C      JP 40180        *
33    40059    23          INC HL
34    40060    AF          XOR A
35    40061    BE          CP (HL)
36    40062    2809        JRZ 40073
```

3) Deplacement et tir canon

```
37    40064    AF          XOR A
38    40065    FDBEB6      CP (IY-74)
```

```
39   40068   2803         JRZ 40073
40   40070   CDB59E       CALL 40629      *
41   40073   2A1AA4       LD HL,(42010)   *
42   40076   010310       LD BC,&1003
43   40079   3E26         LD A,38
44   40081   DDBE04       CP (IX+4)
45   40084   2010         JRNZ 40102
46   40086   DD360400     LD (IX+4),0
47   40090   CD9C9E       CALL 40604      *
48   40093   2130C7       LD HL,50992
49   40096   221AA4       LD (42010),HL
50   40099   C3E89C       JP 40168        *
51   40102   DD3404       INC (IX+4)
52   40105   23           INC HL
53   40106   23           INC HL
54   40107   221AA4       LD (42010),HL   *
55   40110   2B           DEC HL
56   40111   2B           DEC HL
57   40112   CD9C9E       CALL 40604      *
58   40115   C3E89C       JP 40168        *
59   40118   AF           XOR A
60   40119   FDBEB6       CP (IY-74)
61   40122   28303        JRZ 40127
62   40124   CDB59E       CALL 40629      *
63   40127   2A1AA4       LD HL,(42010)   *
64   40130   010310       LD BC,&1003
65   40133   AF           XOR A
66   40134   DDBE04       CP (IX+4)
67   40137   2010         JRNZ 40155
68   40139   DD360426     LD (IX+4),38
69   40143   CD9C9E       CALL 40604      *
70   40146   217CC7       LD HL,51068
71   40149   221AA4       LD (42010),HL   *
72   40152   CDEB9C       JP 40171        *
73   40155   DD3504       DEC (IX+4)
74   40158   2B           DEC HL
75   40159   2B           DEC HL
76   40160   221AA4       LD (42010),HL   *
77   40163   23           INC HL
78   40164   23           INC HL
79   40165   CD9C9E       CALL 40604      *
80   40168   2A1AA4       LD HL,(42010)   *
81   40171   010310       LD BC,&1003
```

```
82   40174   116DA4      LD DE,42093     *
83   40177   CD819E      CALL 40577      *
```

4) Deplacement asteroide

```
84   40180   1800        JR 0
85   40182   2A16A4      LD HL,(42006)   *
86   40185   115DA4      LD DE,42077     *
87   40188   010208      LD BC,&0802
88   40191   CD819E      CALL 40577      *
89   40194   2A16A4      LD HL,(42006)   *
90   40197   01FF17      LD BC,6143
91   40200   09          ADD HL,BC
92   40201   3004        JRNC 40207
93   40203   0150C0      LD BC,-16304
94   40206   09          ADD HL,BC
95   40207   010208      LD BC,&0802
96   40210   115DA4      LD DE,42077     *
97   40213   2216A4      LD (42006),HL   *
98   40216   CD819E      CALL 40577      *
99   40219   2A1AA4      LD HL,(42010)   *
100  40222   CD429E      CALL 40154      *
101  40225   2803        JRZ 40230
102  40227   C3D29E      JP 40658
```

5) Deplacement missile

```
103  40230   182D        JR 40277
104  40232   2A14A4      LD HL,(42004)   *
105  40235   010107      LD BC,&0701
106  40238   1155A4      LD DE,42069     *
107  40241   CD819E      CALL 40577      *
108  40244   2A14A4      LD HL,(42004)   *
109  40247   015000      LD BC,80
110  40250   ED42        SBC HL,BC
111  40252   3EBF        LD A,&BF
112  40254   BC          CP H
113  40255   2817        JRZ 40280
114  40257   2214A4      LD (42004),HL   *
115  40260   1155A4      LD DE,42069     *
116  40263   010107      LD BC,&0701
117  40266   CD819E      CALL 40577      *
118  40269   2A14A4      LD HL,(42004)   *
```

```
119  40272   CD539E      CALL 40531      *
120  40275   200F        JRNZ 40292
121  40277   C3CB9D      JP 40395        *
122  40280   AF          XOR A
123  40281   DDBE00      CP (IX+0)
124  40284   C8          RET Z
125  40285   FD36B62D    LD (IY-74),&2D
126  40289   C3CB9D      JP 40395        *
```

6) Collision missile

```
127  40292   FD36B62D    LD (IY-74),&2D
128  40296   ED4B10A4    LD BC,(42000)   *
129  40300   CD619E      CALL 40545      *
130  40303   EB          EX HL,DE
131  40304   201A        JRNZ 40332
132  40306   AF          XOR A
133  40307   FDBE7F      CP (IY+127)
134  40310   2803        JRZ 40315
135  40312   DD3401      INC (IX+1)
136  40315   DD3401      INC (IX+1)
137  40318   2A10A4      LD HL,(42000)   *
138  40321   CDF79E      CALL 40695      *
139  40324   C8          RET Z
140  40325   FD366716    LD(IY+103),&16
141  40329   C3CB9D      JP 40395        *
142  40332   ED4B12A4    LD BC,(42002)   *
143  40336   CD619E      CALL 40545      *
144  40339   EB          EX HL,DE
145  40340   201A        JRNZ 40368
146  40342   AF          XOR A
147  40343   FDBE67      CP (IY+103)
148  40346   2803        JRZ 40351
149  40348   DD3401      INC (IX+1)
150  40351   DD3401      INC (IX+1)
151  40354   2A12A4      LD HL,(42002)   *
152  40357   CDF79E      CALL 40695      *
153  40360   C8          RET Z
154  40361   FD367F1D    LD(IY+127),&1D
155  40365   C3CB9D      JP 40395        *
156  40368   3E44        LD A,&44
157  40370   BE          CP (HL)
158  40371   200B        JRNZ 40384
```

```
159  40373  54          LD D,H
160  40374  5D          LD E,L
161  40375  7C          LD A,H
162  40376  C638        ADD A,&38
163  40378  67          LD H,A
164  40379  AF          XOR A
165  40380  BE          CP (HL)
166  40381  EB          EX HL,DE
167  40382  2807        JRZ 40391
168  40384  FD368430    LD(IY-124),&30
169  40388  2A16A4      LD HL,(42006)   *
170  40391  CDF79E      CALL 40695      *
171  40394  C8          RET Z
```

7) Deplacement fusees

```
172  40395  2A10A4      LD HL,(42000)   *
173  40398  3EF0        LD A,&F0
174  40400  BD          CP L
175  40401  283E        JRZ 40465
176  40403  2B          DEC HL
177  40404  2210A4      LD (42000),HL   *
178  40407  1800        JR 0
179  40409  23          INC HL
180  40410  111DA4      LD DE,42013     *
181  40413  010407      LD BC,&0704
182  40416  CD819E      CALL 40577      *
183  40419  2A10A4      LD HL,(42000)   *
184  40422  111DA4      LD DE,42013     *
185  40425  010407      LD BC,&0704
186  40428  CD819E      CALL 40577      *
187  40431  1800        JR 0
188  40433  2A12A4      LD HL,(42002)   *
189  40436  23          INC HL
190  40437  2212A4      LD (42002),HL   *
191  40440  2B          DEC HL
192  40441  1139A4      LD DE,42041     *
193  40444  010407      LD BC,&0704
194  40447  CD819E      CALL 40577      *
195  40450  2A12A4      LD HL,(42002)   *
196  40453  1139A4      LD DE,42041     *
197  40456  010407      LD BC,&0704
198  40459  CD819E      CALL 40577      *
```

```
199  40462   C3489C      JP 40008          *
```

8) reinitialisation fusees

```
200  40465   FD366700    LD(IY+103),0
201  40469   FD367F00    LD(IY+127),0
202  40473   2A10A4      LD HL,(42000)     *
203  40476   010407      LD BC,&0704
204  40479   CD9C9E      CALL 40604        *
205  40482   2A12A4      LD HL,(42002)     *
206  40485   010407      LD BC,&0704
207  40488   CD9C9E      CALL 40604        *
208  40491   21DCC1      LD HL,49628
209  40494   2210A4      LD (42000),HL     *
210  40497   010407      LD BC,&0704
211  40500   111DA4      LD DE,42013       *
212  40503   CD819E      CALL 40577        *
213  40506   2130C2      LD HL,49712
214  40509   2212A4      LD (42002),HL     *
215  40512   18C3        JR 40453
```

Sous-programme 1

```
216  40514   015000      LD BC,80
217  40517   AF          XOR A
218  40518   BE          CP (HL)
219  40519   C0          RET NZ
220  40520   23          INC HL
221  40521   23          INC HL
222  40522   BE          CP (HL)
223  40523   C0          RET NZ
224  40524   09          ADD HL,BC
225  40525   BE          CP (HL)
226  40526   C0          RET NZ
227  40527   2B          DEC HL
228  40528   2B          DEC HL
229  40529   BE          CP (HL)
230  40530   C9          RET
```

Sous-programme 2

```
231  40531   3444        LD A,&44
232  40533   BE          CP (HL)
```

```
233   40534   C0          RET NZ
234   40535   44          LD B,H
235   40536   3E20        LD A,32
236   40538   84          ADD A,H
237   40539   67          LD H,A
238   40540   3EEE        LD A,&EE
239   40542   BE          CP (HL)
240   40543   60          LD H,B
241   40544   C9          RET
```

Sous-programme 3

```
242   40545   54          LD D,H
243   40546   5D          LD E,L
244   40547   79          LD A,C
245   40548   BD          CP L
246   40549   2003        JRNZ 40554
247   40551   78          LD A,B
248   40552   BC          CP H
249   40553   C8          RET Z
250   40554   3           INC BC
251   40555   79          LD A,C
252   40556   BD          CP L
253   40557   2003        JRNZ 40562
254   40559   78          LD A,B
255   40560   BC          CP H
256   40561   C8          RET Z
257   40562   3           INC BC
258   40563   79          LD A,C
259   40564   BD          CP L
260   40565   2003        JRNZ 40570
261   40567   78          LD A,B
262   40568   BC          CP H
263   40569   C8          RET Z
264   40570   3           INC BC
265   40571   79          LD A,C
266   40572   BD          CP L
267   40573   C0          RET NZ
268   40574   78          LD A,B
269   40575   BC          CP H
270   40576   C9          RET
```

Sous-programme 4

```
271  40577   F3          DI
272  40578   C5          PUSH BC
273  40579   41          LD B,C
274  40580   E5          PUSH HL
275  40581   1A          LD A,(DE)
276  40582   AE          XOR (HL)
277  40583   77          LD (HL),A
278  40584   23          INC HL
279  40585   13          INC DE
280  40586   10F9        DJNZ 40581
281  40588   E1          POP HL
282  40589   7C          LD A,H
283  40590   C608        ADD A,8
284  40592   67          LD H,A
285  40593   3004        JRNC 40599
286  40595   0150C0      LD BC,-16304
287  40598   09          ADD HL,BC
288  40599   C1          POP BC
289  40600   10E8        DJNZ 40578
290  40602   FB          EI
291  40603   C9          RET
```

Sous-programme 5

```
292  40604   F3          DI
293  40605   C5          PUSH BC
294  40606   41          LD B,C
295  40607   E5          PUSH HL
296  40608   3600        LD (HL),0
297  40610   23          INC HL
298  40611   10FB        DJNZ 40608
299  40613   E1          POP HL
300  40614   7C          LD A,H
301  40615   C608        ADD A,8
302  40617   67          LD H,A
303  40618   3004        JRNC 40624
304  40620   0150C0      LD BC,-16304
305  40623   09          ADD HL,BC
306  40624   C1          POP BC
307  40625   10EA        DJNZ 40605
308  40627   FB          EI
```

```
309  40628   C9          RET

     Sous-programme 6

310  40629   FD36B600    LD(IY-74),00
311  40633   DD3500      DEC (IX+0)
312  40636   2A1AA4      LD HL,(42010)   *
313  40639   23          INC HL
314  40640   015000      LD BC,80
315  40643   ED42        SBC HL,BC
316  40645   2214A4      LD (42004),HL   *
317  40648   1155A4      LD DE,42069     *
318  40651   010107      LD BC,&0701
319  40654   CD819E      CALL 40577      *
320  40657   C9          RET

     Sous-programme 7

321  40658   2A16A4      LD HL,(42006)   *
322  40661   010208      LD BC,&0802
323  40664   CD9C9E      CALL 40604      *
324  40667   2A1AA4      LD HL,(42010)   *
325  40670   010310      LD BC,&1003
326  40673   CD9C9E      CALL 40604      *
327  40676   2A1AA4      LD HL,(42010)   *
328  40679   01040C      LD BC,&0C04
329  40682   119DA4      LD DE,42141     *
330  40685   CD819E      CALL 40577      *
331  40688   21CDA4      LD HL,42189     *
332  40691   CDAABC      CALL &BCAA
333  40694   C9          RET

     Sous-programme 8

334  40695   2214A4      LD (42004),HL   *
335  40698   010408      LD BC,&0804
336  40701   CD9C9E      CALL 40604      *
337  40704   2A14A4      LD HL,(42004)   *
338  40707   01040C      LD BC,&0C04
339  40710   119DA4      LD DE,42141     *
340  40713   CD819E      CALL 40577      *
341  40716   21CDA4      LD HL,42189     *
342  40719   DDE5        PUSH IX
```

```
343  40721   CDAABC      CALL &BCAA
344  40724   DDE1        POP IX
345  40726   214B9C      LD HL,40011     *
346  40729   36CD        LD (HL),&CD
347  40731   23          INC HL
348  40732   3626        LD (HL),&26
349  40734   23          INC HL
350  40735   369F        LD (HL),&9F
351  40737   AF          XOR A
352  40738   DDBE00      CP (IX+0)
353  40741   C9          RET
```

Sous-programme 9

```
354  40742   2A14A4      LD HL,(42004)   *
355  40745   01040C      LD BC,&0C04
356  40748   CD9C9E      CALL 40604      *
357  40751   214B9C      LD HL,40011     *
358  40754   3600        LD (HL),0
359  40756   23          INC HL
360  40757   3600        LD (HL),0
361  40759   23          INC HL
362  40760   3600        LD (HL),0
363  40762   C9          RET
```

```
42000 Adresse fusee 1
42002 Adresse fusee 2
42004 Adresse missile
42006 Adresse asteroide
42008 Compteur missile
42009 Compteur de points
42010 Adresse canon
42012 Compteur canon
42013 Dessin fusee 1
42041 Dessin fusee 2
42069 Dessin missile
42077 Dessin asteroide
42093 Dessin canon
42141 Dessin explosion
42189 Queue sonore
```

Programme de chargement BASIC :

```
10 MEMORY 39999
20 CLS:PRINT"CHARGEMENT EN COURS"
30 '
40 '
50 AD=40000
60 FOR I=0 TO 75
70 READ A$,B$:V=0
80 FOR H=1 TO LEN(A$)-1 STEP 2
90 X=VAL("&"+MID$(A$,H,2)):V=V+X
100 POKE AD,X:AD=AD+1:IF AD=40763 THEN A
D=42000
110 NEXT H
120 IF V<>VAL("&"+B$) THEN PRINT"ERREUR
EN LIGNE ";160+I*10:END
130 NEXT I
140 '
150 '
160 DATA DD2118A4FD21719DCD19BD0000,589
170 DATA 0021EBB4AFBE2320113E02BE20,49F
180 DATA 0CAFFDBEB62806CDB59EC3F49C,7CD
190 DATA 3E03BE284A3E01BE284E2BBEC8,495
200 DATA 3CBE2803C3F49C23AFBE2809AF,5E8
210 DATA FDBEB62803CDB59E2A1AA40103,5A8
220 DATA 103E26DDBE042010DD360400CD,427
230 DATA 9C9E2130C7221AA4C3E89CDD34,68A
240 DATA 042323221AA42B2BCD9C9EC3E8,532
250 DATA 9CAFFDBEB62803CDB59E2A1AA4,6EF
260 DATA 010310AFDDBE042010DD360426,3CF
270 DATA CD9C9E217CC7221AA4C3EB9CDD,772
280 DATA 35042B2B221AA42323CD9C9E2A,3E6
290 DATA 1AA4010310116DA4CD819E1800,3F8
300 DATA 2A16A4115DA4010208CD819E2A,417
310 DATA 16A401FF17093004015OC00901,329
320 DATA 0208115DA42216A4CD819E2A1A,428
330 DATA A4CD429E2803C3D29E182D2A14,532
340 DATA A40101071155A4CD819E2A14A4,485
350 DATA 015000ED423EBFBC28172214A4,452
360 DATA 1155A4010107CD819E2A14A4CD,4AE
370 DATA 539E200FC3CB9DAFDDBE00C8FD,75A
380 DATA 36B62DC3CB9DFD36B6,52D
390 DATA 2DED4B10A4CD619EEB,4D0
```

```
400 DATA 201AAFFDBE7F2803DD,42B
410 DATA 3401DD34012A10A4CDF79EC8FD,64C
420 DATA 366716C3CB9DED4B12A4CD619E,698
430 DATA EB201AAFFDBE672803DD3401DD,610
440 DATA 34012A12A4CDF79EC8FD367F1D,60E
450 DATA C3CB9D3E44BE200B545D7CC638,5C1
460 DATA 67AFBEEB2807FD3684302A16A4,5B9
470 DATA CDF79EC82A10A43EF0BD283E2B,684
480 DATA 2210A4180023111DA4010407CD,2BC
490 DATA 819E2A10A4111DA4010407CD81,429
500 DATA 9E18002A12A4232212A42B1139,306
510 DATA A4010407CD819E2A12A41139A4,46A
520 DATA 010407CD819EC3489CFD366700,539
530 DATA FD367F002A10A4010407CD9C9E,4A3
540 DATA 2A12A4010407CD9C9E21DCC122,4D3
550 DATA 10A4010407111DA4CD819E2130,3CF
560 DATA C22212A418C3015000AFBEC023,516
570 DATA 23BEC009BEC02B2BBEC93E44BE,645
580 DATA C0443E2084673EEEBE60C9545D,611
590 DATA 79BD200378BCC80379BD200378,529
600 DATA BCC80379BD200378BCC80379BD,615
610 DATA C078BCC9F3C541E51AAE772313,710
620 DATA 10F9E17CC60867300401500009,4E9
630 DATA C110E8FBC9F3C541E536002310,6C4
640 DATA FBE17CC60867300401500C09C1,59C
650 DATA 10EAFBC9FD36B600DD35002A1A,5FD
660 DATA A423015000ED422214A41155A4,42B
670 DATA 010107CD819EC92A16A4010208,3AD
680 DATA CD9C9E2A1AA4010310CD9C9E2A,534
690 DATA 1AA401040C119DA4CD819E21CD,4FB
700 DATA A4CDAABCC92214A4010408CD9C,5F0
710 DATA 9E2A14A401040C119DA4CD819E,4CF
720 DATA 21CDA4DDE5CDAABCDDE1214B9C,84D
730 DATA 36CD23362623369FAFDDBE00C9,58D
740 DATA 2A14A401040CCD9C9E214B9C36,438
750 DATA 002336002336 00C9,17B
760 DATA dcc130C200004ec0140030c700,4A8
770 DATA 112333ff11e8674c33f9efccff,6F8
780 DATA f92d8833f9efcc11e8674c1123,675
790 DATA 33ffffcc4c88236e7188337ff9,706
800 DATA cc114bf9ff337ff9cc236e7188,721
810 DATA ffcc4c884444ee44eeeeeeaa33,800
820 DATA cc44ff77eeffbb6644eeeeff77,92A
```

```
830 DATA 11cc00660000660000660000000f,21E
840 DATA 0000660000ff00016f0800ff00,2DC
850 DATA 009f0004ff021c96837890e168,52A
860 DATA 996148ff211d99a90899018899,584
870 DATA 001144118866773399cc33f6fb,587
880 DATA 8811f1f32233fbf8ccfff9f3ff,97B
890 DATA 11f2f98833f6fd8833bbbbcc66,80D
900 DATA 1188ee8855001101010000001e,295
910 DATA 009600,96
920 '
930 '
940 MODE 1:INK 0,0:BORDER 0
950 '
960 FOR I=1 TO 20:POKE INT(RND*1840)+511
99,136:POKE INT(RND*1840)+51199,128:POKE
 INT(RND*1840)+51199,4:NEXT
970 '
980 AD=42093
990 A=50992:B=65328:GOSUB 1010
1000 A=51072:B=65408:GOSUB 1010:GOTO 107
0
1010 FOR I=A TO B STEP 2048
1020 FOR H=I TO I+2
1030 POKE H,PEEK(AD):AD=AD+1
1040 NEXT H,I
1050 RETURN
1060 '
1070 SYMBOL 255,60,79,126,251,100,238,24
7,28
1080 PEN 3:LOCATE 40,1:PRINT CHR$(255)
1090 '
1100 '
1110 AD=42013
1120 FOR I=49628 TO 61916 STEP 2048
1130 FOR H=I TO I+3:POKE H,PEEK(AD)
1140 AD=AD+1:NEXT H,I
1150 '
1160 AD=42041
1170 FOR i=49712 TO 62000 STEP 2048
1180 FOR H=I TO I+3:POKE H,PEEK(AD)
1190 AD=AD+1:NEXT H,I
1200 '
1210 SYMBOL 255,0,79,126,251,100,238,119
```

```
,0
1220 FOR I=1 TO 6:X=INT(RND*36)+3:Y=INT(
RND*15)+4
1230 LOCATE X,Y:PEN 3:PRINT CHR$(255):NE
XT I:PEN 1
1240 '
1250 ENV 1,1,15,1,1,0,10,50,-1,10
1260 '
1270 CALL 40000
1280 LOCATE 1,1:PRINT"VOTRE SCORE: ";PEE
K(42009)
1290 GOTO 1290
```

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 0040

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 0039

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

ROBOTS - CONSTRUCTION, PROGRAMMATION *par Fernand Esteves,*
400 pages, Réf. 0130

ALGORITHMES *par Pierre Beaufils et Wolfram Luther,*
296 pages, Réf. 0149

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 0129

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

TECHNIQUES DE PROGRAMMATION EN BASIC *par S. Crosmarie, M. Perron et D. Philippine*
152 pages, Réf. 0124

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 0030

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

INTRODUCTION A C *par Bruce Hunter,*
312 pages, Réf. 0092

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

AMSTRAD, GUIDE DU GRAPHISME *par James Wynford,*
208 pages, Réf. 0141

AMSTRAD CP/M 2.2 *par Anatole D'Hardencourt,*
248 pages, Réf. 0156

AMSTRAD ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0167

AMSTRAD MULTIPLAN *de Microsoft,*
496 pages, Réf. 1111

AMSTRAD, CRÉER DE NOUVELLES INSTRUCTIONS
par Jean-Claude Despoine,
144 pages, Réf. 0176

AMSTRAD ASTROCALC
par Gérard Blanc et Philippe Destrebecq,
168 pages, Réf. 0162

AMSTRAD, GRAPHISME EN TROIS DIMENSIONS
par Thomas Lachand-Robert,
240 pages, Réf. 0157

AMSTRAD, GUIDE DU BASIC ET DE L'AMSDOS
par Jean-Louis Gréco et Michel Laurent.
288 pages, Réf. 0159

AMSTRAD CP/M PLUS *par Anatole D'Hardancourt,*
208 pages, Réf. 0184

AMSTRAD, GAGNEZ AUX COURSES
par Jean-Claude Despoine,
112 pages, Réf. 0197

AMSTRAD LOCOSCRIPT *par Bernard Le Dû,*
144 pages, Réf. 0202

AMSTRAD, MISE AU POINT DES PROGRAMMES BASIC
par Claude Vivier et Yvon Jacob,
152 pages, Réf. 0166

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

INTRODUCTION A MAC PASCAL *par Pierre Le Beux,*
416 pages, Réf. 0145

MACINTOSH POUR LA PRESSE, L'EDITION ET LA PUBLICITE *par Bernard Le Du,*
160 pages, Réf. 0173

INTRODUCTION A MAC BASIC
par Pierre Lebeux,
440 pages, Réf. 0140

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DE PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MO 5, STATIQUE, DYNAMIQUE, ELECTRONIQUE, PROGRAMMES DE PHYSIQUE EN BASIC
par Beaufils, Lamarche et Muggianu,
240 pages, Réf. 0148

MO 5, PROGRAMMES D'ELECTRONIQUE EN BASIC
par Beaufils, Delusurieux, Do, Romanacce,
312 pages, Réf. 0143

MO 5, OPTIQUE, THERMODYNAMIQUE, CHIMIE
par P. Beaufils, M. Lamarche, Y. Muggianu,
224 pages, Réf. 0161

MO 5, PROGRAMMES D'ÉLECTRONIQUE APPLIQUÉE
par Pierre Beaufils et Bernard Desperrier,
192 pages, Réf. 0194

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

MSX, PROGRAMMES EN LANGAGE MACHINE
par Steeve Webb,
112 pages, Réf. 0153

MSX, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
216 pages, Réf. 0144

MSX, GUIDE DU BASIC *par Michel Laurent,*
264 pages, Réf. 0155

MSX, JEUX EN ASSEMBLEUR *par Eric Ravis*
112 pages, Réf. 0170

MSX, ROUTINES GRAPHIQUES EN ASSEMBLEUR
par Steeve Webb
88 pages, Réf. 0154

MSX, TECHNIQUES DE PROGRAMMATION DES JEUX EN ASSEMBLEUR
par Georges Fagot-Barraly,
176 pages, Réf. 0178

MSX ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
157 pages, Réf. 0168

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TO 7 / MO 5, GUIDE DU BASIC
par Jean-Louis Greco et Michel Laurent,
288 pages, Réf. 0158

TO 7 / MO 5, GUIDE DU GRAPHISME
par Michel Lamarche et Yves Muggianu,
240 pages, Réf. 0172

TO 7 / MO 5 ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0169

TO 7 / MO 5, JEUX DE RÉFLEXION
par Georges Fagot-Barraly,
176 pages, Réf. 0201

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 0058

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 0181

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel-Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088 *par François Retoreau,*
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

INTRODUCTION A UNIX *par John D. Halamka,*
240 pages, Réf. 0098

GUIDE DE PRODOS
par Pierre Beaufils et Wolfram Luther,
248 pages, Réf. 0146

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

INTRODUCTION A dBASE III *par Alan Simpson,*
272 pages, Réf. 0131

LOTUS 1-2-3 POUR L'ENTREPRISE
par Dominique Helle et Guy Boussand,
256 pages, Réf. 0147

LOTUS 1-2-3 PROGRAMMATION DES MACRO-COMMANDES *par Goulven Habasque,*
144 pages, Réf. 0150 F

INTRODUCTION A WORDSTAR 2000
par David Kolodney et Thomas Blackadar,
256 pages, Réf. 0160

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
352 pages, Réf. 0115

ALGORITHMES *par Pierre Beaufils, et Wolfram Luther,*
296 pages, Réf. 0149

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 61 80 2-0
Telex : 08588163

Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 12 mai 1986 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 2e trimestre 1986 - Imprimeur n° 2025

Cet ouvrage présente une vingtaine de routines en assembleur destinées aux Amstrad CPC 464, 664 et 6128. Leur description est accompagnée d'une étude détaillée des principes de mise en œuvre de sous-programmes assembleurs : détournement de vecteurs d'appel de routines internes ou de vecteurs d'indirection, programmation des interruptions, écriture en mémoire écran, lecture en ROM et utilisation des instructions de RESTART, utilisation du générateur de caractères, stockage des programmes en RAM, variables système, organisation des tableaux. Chaque routine est accompagnée d'un programme de chargement et d'un programme de démonstration en BASIC.

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/